A Formal Approach to Personalisable, Adaptive
Hyperlink-Based Systems

By

James Francis Ohene-Djan

This thesis is submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy
in the
Department of Mathematical and Computing Sciences
Goldsmiths College
University of London
New Cross, London SE14 6NW, UK.

© 2002 James Francis Ohene-Djan
Abstract

The attraction of hyperlink-based interaction as a model for information retrieval has long been recognised and has increased in popularity with the mainstream emergence of large-scale hypermedia systems such as the World-Wide Web (WWW). For hypermedia systems to realise their full potential, researchers have postulated that such systems should exhibit sophisticated, knowledge-based personalisation and adaptation (P&A) features, without which users’ information retrieval goals are less likely to be achieved. As a result of these postulations, personalisable, adaptive hyperlink-based systems (PA-HLBSs) have arisen as a new topic of hypermedia research.

This dissertation contributes a novel abstract approach to the formal characterisation of the interaction process which takes place between the user of a hyperlink-based system (HLBS) and the system itself.

This research addresses the issue of how hyperlink-based systems can be endowed with features which enable the personalisation and adaptation of the interaction process. This research also addresses the specific issue of how to characterise precisely the emergent properties of HLBSs and thereby make possible a systematic, principled and exhaustive elicitation of the space of possible P&A actions.

The approach is unique in formally modelling a rich set of abstract user-initiated P&A actions which enable individual users to come closer to satisfying their specific, and often dynamic, information retrieval goals. Furthermore, the model indicates how system-initiated P&A actions fit cohesively and non-disruptively with user-initiated ones.

The model proposed is descriptive, rather than prescriptive, and is cast at a level of abstraction above that of concrete systems exploring current technologies. The model aims to be the foundation for a systematic investigation of the nature, scope and effects of user and system-initiated tailoring actions on HLBSs for information retrieval. Such an approach, it is hoped, will allow for user and system-initiated P&A actions to be studied with greater conceptual clarity than is possible with technology-driven experimentation.

The dissertation also contains a brief overview of PAS, a personalisable HLBS which instantiates the major aspects of the proposed model, thereby substantiating the claim that the abstract approach taken allows not only for a greater understanding of what personalisation and adaptivity means in the context of HLBSs, but also how the model may aid the design of such systems.
Dedication

This dissertation is dedicated to my mother, Mrs B. Ohene-Djan and to the memory of my father, Dr. I.L. Ohene-Djan. I am forever grateful for the love, support and guidance you have given me throughout my life. God bless you both.

Mum, thanks for buying me my first computer way back in 1989. It was expensive then and would be expensive now but I hope this dissertation goes some way to repaying your investment.
Acknowledgments

I acknowledge with deepest and sincerest gratitude the role played by my supervisor Prof. Nelson Stephens in the research reported in this dissertation. The guidance, patience and support you have given me has been invaluable.

Special thanks are reserved for Dr. Alvaro Fernandes, my second supervisor during much of this research and a valued friend.

Eu gostaria de agradecer toda vossa dedicação à minha pessoa durante os últimos quatro anos. Gostaria também de enfatizar, que vosso modelo de profissionalismo em uma insaciável busca da completa perfeição é e será o eterno guia durante minha carreira profissional e pessoal. Concluindo, tê-lo como professor foi o melhor fato ocorrido durante minha vida educacional. A vossa pessoa, muitissimo obrigado por tudo.

I am also grateful to Dr. Chris Fox and Dr. Rob Hierons for their technical assistance during crucial periods of the research conducted. Their unfaltering patience and their dedication have been warmly appreciated.

I would also like to thank Stuart Ross, Dr. Josh Newton, Rachel Shipsey and Rowan Davis for the advice, support and inspiration that they have provided me with during my long journey.

The Department of Mathematical and Computing Sciences of Goldsmiths College, University of London is an excellent environment for teaching and research. I acknowledge with gratitude the help and support provided by its members and thank them for their dedication and competence.

The research reported in this dissertation was funded by a Goldsmiths College bursary. I wish to thank the College for awarding me this bursary and supporting me whilst conducting this research.

I would like to thank Kate Hargreaves for all her love, support and understanding during what has sometimes been a difficult journey.

Finally, but far from least, my sincere thanks to my family and my many friends for their emotional support, patience and understanding. You were all fantastic.
Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

James Ohene-Djan
Department of Mathematical and Computing Sciences
Goldsmiths College, London University
July 12, 2002
## Contents

Abstract ................................................................. i

Dedication ................................................................... ii

Acknowledgments ......................................................... iii

Declaration ........................................................................ iv

List of Figures ............................................................... xiv

1 Introduction ................................................................. 1

1.1 Background .......................................................... 1

1.2 Motivation and Challenges ......................................... 4

1.2.1 Motivation ........................................................ 4

1.2.2 Challenges ....................................................... 8

1.3 Technical Aims of Research ........................................ 9

1.3.1 The Central Question ........................................ 9

1.3.2 Derived Problems ............................................... 9

1.4 Research Contribution .............................................. 11

1.4.1 A Formal Model ............................................... 12

1.4.2 PAS: A Personalisable Hyperlink-Based System ............ 15

1.5 Thesis Structure ....................................................... 15

2 Personalisable, Adaptive HLBSs Research Issues .................. 17

2.1 Understanding Hyperlink-Based Systems ....................... 17

2.1.1 The Cognitive Basis of Hyperlink-Based Systems ............ 17

2.1.2 Defining Hyperlink-Based Systems .......................... 19

2.2 Historical Context .................................................. 21

2.2.1 Origins of HLBSs Research .................................... 21

2.2.2 Models of HLBSs ............................................... 24

2.2.3 HLBSs Interaction Issues ...................................... 26
2.3 Personalisable, Adaptive HLBSs ........................................ 27
  2.3.1 Motivations for Personalisation Requests ...................... 28
  2.3.2 Techniques Used to Implement Personalisation Requests ........ 31
  2.3.3 User Modelling in Personalisable, Adaptive HLBSs .............. 34
  2.3.4 Applications of Personalisable, Adaptive HLBSs .................. 37
2.4 Summary Discussion .................................................. 38

3 A Survey of Hyperlink-Based Modelling Issues ......................... 40
  3.1 An Introduction to Hyperlink-Based Modelling ...................... 40
    3.1.1 Modelling Hyperpages ......................................... 41
    3.1.2 Modelling Hyperlinks ......................................... 42
    3.1.3 Modelling Hyper-networks .................................... 45
    3.1.4 Requirements for HLBSs Models ................................ 46
  3.2 Modelling Instantiations of HLBSs .................................. 49
    3.2.1 The Hypertext Abstract Machine (HAM) ......................... 50
    3.2.2 The Dexter Hypertext Reference Model ......................... 52
  3.3 Modelling the Integration of HLBSs ................................. 56
    3.3.1 An Introduction to OHLBSs Models ............................. 56
    3.3.2 Architectural Models of OHLBSs ............................... 59
    3.3.3 The Flag Taxonomy for OHLBSs ................................ 62
  3.4 Modelling User Interaction with HLBSs .............................. 64
    3.4.1 HLBSs Modelling using Natural Language Generation ............ 64
    3.4.2 A Dexter-Based Reference Model for Adaptive Hypermedia ........ 67
    3.4.3 The Standard Reference Model ................................ 67
    3.4.4 SMIL: A Personalisable Multimedia Integration Language ........ 68
  3.5 An Introduction to the Model for Personalisable, Adaptive Hyperlink-Based Interaction ........................................... 70
    3.5.1 Architectural Assumptions .................................... 70
    3.5.2 Technical Approach ........................................... 72

4 The H-Region: Modelling Non-Personalisable, Non-Adaptive Hyperlink-Based Interaction .............................................. 74
  4.1 The H-Region: An Introduction ..................................... 74
  4.2 The H-Region: A Conceptual Framework ............................. 75
  4.3 The H-Region: Dynamics of the H-Region ............................ 76
4.4 The H-Region: Formal Elements of the H-Region ......................... 79
  4.4.1 An Introduction to Context Free Grammars ......................... 79
  4.4.2 A Formal Grammar for the Specification of Hyperpages ........... 81
4.5 Formal Semantics of Hyperpage Specifications .......................... 85
  4.5.1 Hyperpage Abstract Machine .................................. 85
4.6 Opportunities Afforded by the H-Region ................................ 86
4.7 What Next?: A Model of User-Initiated Personalisable Hyperlink-Based Interaction .............................................. 87

5 The P-Region: Modelling User-Initiated Personalisable Hyperlink-Based Interaction 89
  5.1 The P-Region: An Introduction ...................................... 89
  5.2 The P-Region: A Conceptual Framework ............................... 91
  5.3 The P-Region: Dynamics of the P-Region ............................ 92
  5.4 The P-Region: Formal Elements of the P-Region ..................... 93
    5.4.1 A Formal Grammar for the Specification of User Requests ....... 95
    5.4.2 A Formal Grammar for the Specification of Hyperpage Annotations 96
    5.4.3 A Formal Grammar for the Specification of Personalisation Requests 99
    5.4.4 Examples of Personalisation Requests ........................... 103
  5.5 Formal Semantics of Personalisation Requests ........................ 105
    5.5.1 Hyperpage and Hyperpage Annotation Relational Schemas ....... 106
    5.5.2 Preliminary Definitions ..................................... 110
    5.5.3 Semantics .................................................. 112
    5.5.4 Examples of Personalisation Request Programs .................. 136
  5.6 Opportunities Afforded by the P-Region ................................ 139
  5.7 What Next?: A Model of System-Driven Personalisable Hyperlink-Based Interaction .............................................. 142

6 The A-Region: Future Work on Modelling System-Initiated Personalisable Hyperlink-Based Interaction 143
  6.1 The A-Region: An Introduction ...................................... 143
  6.2 The A-Region: A Conceptual Framework ............................... 144
  6.3 The A-Region: Dynamics of the A-Region ............................ 147
  6.4 A Proposed Architecture for the Adaptive Function ................ 149
    6.4.1 An Active Rule Program ...................................... 149
8.5 Comparisons with Generalised Mark-up Languages ................. 194
8.5.1 Comparisons with XML and SMIL .......................... 195
8.6 Summary of Related Work ..................................... 196

9 Contributions, Future Directions and Conclusions ...................... 198
9.1 Contributions .................................................. 198
9.1.1 Motivation Revisited ......................................... 199
9.1.2 Challenges Revisited ......................................... 200
9.1.3 Technical Aims Revisited ...................................... 200
9.1.4 Proposed Solution ........................................... 201
9.1.5 Summary of Contributions .................................... 202
9.1.6 Applications for Research .................................... 206
9.2 Future Directions for Research .................................. 208
9.2.1 Structural Hyperlink-Based Computing ...................... 208
9.2.2 Defining the PA-HLBSs Research Space ...................... 210
9.2.3 HLBSs and Active Databases .................................. 210
9.2.4 Principles for Personalisation and Adaptation ............... 211
9.2.5 Large Scale Empirical Research .............................. 212
9.3 Research Conclusions ........................................... 212

Appendices ............................................................... 213

A Formal Semantics of Personalisation Requests using Set-theoretic Expressions ........................................... 214
A.1 The Meaning Function ............................................. 214
A.1.1 Meaning: Personalisation Request ............................ 215
A.1.2 Extending the Meaning Function ............................... 228
A.1.3 Functions for the Insertion and Deletion of Notes ........... 229

B Procedural Design: The H-Region .................................. 236

C Procedural Design: The P-Region .................................. 243

D A Prolog Parser for Hyperpages, Hyperpage Annotations and Personalisation Requests ................................. 253
D.1 Parser defining DCG grammars for personalisation requests, hyperpages and hyperpage annotations .................. 253
D.2 Loader for lexical, syntactic and semantic analysers in PAS . . . . . . . . . 267
D.3 Sample Personalisation Request Text File . . . . . . . . . . . . . . . . . . . 290
D.4 Personalisation Requests Program Run . . . . . . . . . . . . . . . . . . . . . 291

E PAS Front-End Cold Fusion Application Programs 299
E.1 A WWW-based Application for the Dynamic Generation of Hyperpages . . 299
E.2 A WWW-based Application for the Display of Hyperpage Annotations . . 311

F The Relational Data Model and Relational Algebra 313
F.1 Basic Concepts of Relational Data Model . . . . . . . . . . . . . . . . . . . . . 313
   F.1.1 Relational Data Model Definitions . . . . . . . . . . . . . . . . . . . . . . 313
F.2 Relational Algebra . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 315
   F.2.1 Retrieval Operations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 315
   F.2.2 Update Operations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 318

Glossary 319

Bibliography 321
## List of Figures

1.1 Motivation for Personalisation and Adaptation in HLBSs .................. 5

2.1 Process of Reading and Writing using Traditional Linear Media .......... 18
2.2 Process of Reading and Writing using a HLBS ............................. 18
2.3 User Information Goal Motivations for Personalisation ........................ 29
2.4 User History Motivations for Personalisation .............................. 30

3.1 Static and Dynamic Models of Hyperpage Retrieval .......................... 42
3.2 Models of Hyperlink Implementation Strategies .............................. 45
3.3 Hyper-network Information Structures ...................................... 46
3.4 The Layers of the HAM ....................................................... 51
3.5 Layers of the Dexter Model ................................................... 53
3.6 Typical Architecture of a HBMS .............................................. 59
3.7 Typical Architecture of a Link Server ....................................... 60
3.8 Typical Open Hyperbase Architecture ....................................... 61
3.9 Typical Architecture of the WWW ............................................ 62
3.10 The Flag Taxonomy for OHLBSs .............................................. 62
3.11 Examples of Applying the Flag Taxonomy to HLBSs .......................... 63
3.12 System Architecture of HLBSs Incorporating NLG Functionality .......... 66
3.13 A General, Open Architecture for HLBSs .................................... 70

4.1 The H-Region ............................................................... 77
4.2 Expanding the Perform Retrieval Function in Figure 4.1 .................... 77
4.3 H-Region Functions: Signatures and Types ................................... 79
4.4 An EBNF Grammar of User Requests to the H-Region ........................ 81
4.5 An EBNF Grammar of Hyperpages ........................................... 82
4.6 An Example Hyperpage Specification ......................................... 84
4.7 Abstract Machine Instructions for the Hyperpage Composition ............ 85
4.8 An Example program .......................................................... 86

5.1 Superimposing the P-Region onto the H-Region .............................. 92
5.2 P-Region Functions: Signatures and Types ........................................... 94
5.3 An EBNF Grammar of User Requests to the P-Region .......................... 95
5.4 An EBNF Grammar of Hyperpage Annotations (I) .......................... 96
5.5 An EBNF Grammar of Hyperpage Annotations (II) ......................... 97
5.6 An Example of a Hyperpage Annotation .............................. 99
5.7 An EBNF Grammar of Personalisation Requests (I) .................. 100
5.8 An EBNF Grammar of Personalisation Requests (II) .................. 101
5.9 Function: \texttt{Map\_tuple (I)} ......................................................... 129
5.10 Function: \texttt{Map\_tuple (II)} ......................................................... 130
5.11 Function: \texttt{Map\_tuple (III)} ......................................................... 131
5.12 Function: \texttt{Map\_tuple (IV)} ......................................................... 132
5.13 Function: \texttt{Map\_tuple (V)} ......................................................... 133
5.14 Function: \texttt{Map\_tuple (VI)} ......................................................... 134
5.15 Personalisation Program Example 1 .............................................. 136
5.16 Personalisation Program Example 2 .............................................. 136
5.17 Personalisation Program Example 3 .............................................. 137
5.18 Personalisation Program Example 4 .............................................. 137
5.19 Personalisation Program Example 5 .............................................. 138
5.20 Personalisation Program Example 6 .............................................. 138

6.1 Superimposing the A-Region onto the P- and H-Regions .................. 147
6.2 Example Rules .................................................................................. 155

7.1 Architecture of PAS ......................................................................... 163
7.2 Personalisation Request and Generated Token Stream: Example 1 ........ 166
7.3 Personalisation Request and Generated Token Stream: Example 2 ........... 166
7.4 Token Stream for Example 1 .......................................................... 167
7.5 Abstract Syntax Tree of Token Stream: Example 1 ................................ 167
7.6 Token Stream for Example 2 .......................................................... 168
7.7 Abstract Syntax Tree of Token Stream: Example 2 ................................ 168
7.8 Term Representation of Abstract Syntax Tree: Example 1 .................. 169
7.9 Term Representation of Abstract Syntax Tree: Example 2 .................. 169
7.10 Hyper-library Relational Schemas .................................................... 170
7.11 SQL Representation of Example 1 .................................................. 171
7.12 SQL Representation of Example 2 ........................................ 172
7.13 The Back-End Component of PAS ..................................... 173
7.14 The Front-End Component of PAS ..................................... 175

8.1 A Comparison of the Dexter Model and the Abstract Model Proposed ... 183
8.2 Representative Examples of OHLBSs Modelling Research .......... 184
8.3 The Proposed Model Represented using The Flag Taxonomy .......... 187
8.4 Representative Examples of PA-HLBSs ................................. 190

A.1 Function : InsertNote(note) (I) ........................................ 231
A.2 Function : InsertNote(note) (II) ....................................... 232
A.3 Function : InsertNote(note) (III) ..................................... 233
A.4 Function : DeleteNote(note) (I) ....................................... 234
A.5 Function : DeleteNote(note) (II) ..................................... 235
A.6 Function : DeleteNote(note) (III) ..................................... 235

B.1 Procedural Design of the H-Region (1) ................................. 237
B.2 Procedural Design of the H-Region (2) ................................. 238
B.3 Procedural Design of the H-Region (3) ................................. 238
B.4 Procedural Design of the H-Region (4) ................................. 239
B.5 Procedural Design of the H-Region (5) ................................. 239
B.6 Procedural Design of the H-Region (6) ................................. 240
B.7 Procedural Design of the H-Region (7) ................................. 241
B.8 Procedural Design of the H-Region (8) ................................. 241
B.9 Procedural Design of the H-Region (9) ................................. 242

C.1 Basic Design of the P-region (1) ....................................... 244
C.2 Basic Design of the P-region (2) ....................................... 245
C.3 Basic Design of the P-region (3) ....................................... 245
C.4 Basic Design of the P-region (4) ....................................... 246
C.5 Basic Design of the P-region (5) ....................................... 246
C.6 Basic Design of the P-region (6) ....................................... 247
C.7 Basic Design of the P-region (7) ....................................... 247
C.8 Basic Design of the P-region (8) ....................................... 248
C.9 Basic Design of the P-region (9) ....................................... 249
C.10 Basic Design of the P-region (10) .................................... 249
Chapter 1
Introduction

This introduction constitutes a broad overview of all the research results reported in this dissertation. The introduction discusses the background, motivation and challenges for the research and provides a preliminary overview of related work. The central question which the research endeavours to solve is stated together with the methods through which a solution was achieved. The introduction closes with a summary of the contributions made by the research results.

1.1 Background

This dissertation characterises and investigates the formal properties of a class of computer systems referred to as hyperlink-based systems (HLBSs)\(^1\). HLBSs enable the creation, manipulation and examination of a network of information units which may comprise text, graphics, video, animation and/or sound. In this dissertation such a network is referred to as a hyper-network\(^2\). When these information units are rendered, they provide the user with optional links to other information units. Such links can provide context-based, non-linear navigation between information units. Many kinds of HLBSs have been implemented and are generally referred to as hypertext or hypermedia systems. Early accounts of HLBSs research can be found in [Conklin, 1987, Nielsen, 1990]. Surveys of more recent hypermedia concepts and the research that has given rise to them can be found in [Nielsen, 1995, Lennon, 1997, Lowe and Hall, 1999].

Early HLBSs can be viewed as closed, monolithic systems within which a tight coupling of data and user-interface components could be observed. Although such systems served to demonstrate how HLBSs may be realised, questions of interoperability, interchangeability and comparability could not be easily addressed. Furthermore, HLBSs were criticised for their inability to be easily integrated with existing information environments.

A growing awareness of these weaknesses led to research into, among others, two

\(^{1}\)This term, which is adopted throughout the dissertation, is used in preference to hypertext and hypermedia not only because it is more general, but also because it highlights the truly distinctive characteristic of hypertext and hypermedia systems. See subsection 2.1.2 for a fuller explanation.

\(^{2}\)A glossary of terms and acronyms which the reader might find useful to refer to whilst reading this dissertation can be found in the appendix.
classes of HLBSs: *hyperbase management systems* (HBMSs) [Isakowitz et al., 1995, Grønbæk and Trigg, 1996, Wiil and Leggett, 1996], and *link server systems* (LSSs) [Pearl, 1989, Davis et al., 1992b, Rizk and Sauter, 1992, Grønbæk and Trigg, 1996].

These relatively independent classes of HLBSs have been grouped into the general class termed *open hyperlink-based systems* (OHLBS) [Trigg et al., 1992, Wiil and Østerbye, 1994, Wiil and Demeyer, 1996, Wiil, 1997, Wiil, 1999].

OHLBSs research aims to focus attention on how HLBSs may be integrated with various external servers (e.g., data and link servers) [Fountain et al., 1990, Davis et al., 1994, Grønbæk and Trigg, 1996] as well as third party applications [Anderson et al., 1994, Nürnberg et al., 1996]. This research is based on the view that most applications are hyperlink unaware (i.e., have no notion of linking) but may benefit from such functionality.

OHLBSs research has delivered a variety of models including DHM [Grønbæk and Trigg, 1994a, Grønbæk et al., 1997], AHM [Hardman et al., 1993, Hardman et al., 1994] and systems such as Sun’s Link Service [Pearl, 1989], Hyperdisco [Wiil and Leggett, 1996, Wiil and Leggett, 1997], Microcosm [Fountain et al., 1990, Davis et al., 1992a, Davis et al., 1993], Chimera [Anderson et al., 1994, Anderson, 1997], SP3 [Leggett and Schnase, 1994] and HyperStorM [Bapat et al., 1996].

HLBSs have been criticised for the ad-hoc approach often taken towards their engineering and development [Lowe and Hall, 1999]. In particular, the lack of widely accepted abstract principles and models and the failure to adopt an accepted terminology and taxonomy has led senior researchers to question the theoretical basis of much of the research conducted. This opinion was put forcefully in the keynote address at Hypertext’98 [Leggett, 1998].

Although work has been conducted into isolating the functionality which OHLBSs could provide there is still no consensus regarding the functionality that is unique to HLBSs and does not simply arise as a consequence of coupling, say, user-interface, database and link management components in a particular manner. A conceptualisation addressing this issue is fundamental not only to understanding what HLBSs are, but also how their essential characteristics might be exploited to increase their effectiveness for information retrieval. More generally, if research is to be conducted into hyperlink-based interaction, then it is first necessary to clearly identify the aspects which are unique to HLBSs and therefore should be the subject of such research.

Leading OHLBSs researchers challenge [Nürnberg and Leggett, 1997] the premise that the majority of applications can be characterised as hyperlink unaware and that the role of OHLBSs research in devising a core functionality is to enable such applications to become more aware. In fact, an ever increasing number of applications may be described as hyperlink aware [Grauer, 1997].

It is argued [Nürnberg and Leggett, 1997] that although this new generation of applications
are hyperlink aware, structure management (i.e., management of the hyper-network) is still largely ad hoc. Therefore, the focus of OHLBSs research should now fall on devising effective support mechanisms for structure-oriented computing.

In this dissertation a model is proposed which aims to address the issue of how to formally model a tailorable hyperlink structure which can effectively support a structure-oriented hyperlink computing environment.

Another significant limitation in many current HLBSs is their inability to address users’ information retrieval needs on an individualised basis [Brusilovsky, 1996a, Beaumont, 1994, Gonschorek and Herzog, 1995, Brusilovsky, 1996b]. Most HLBSs provide distinct users with the same information retrieval possibilities and presents that information in the same manner, disregarding user-specific information goals and individual histories. This has resulted in a class of HLBS termed personalisable, adaptive hyperlink-based systems (PA-HLBSs). Although many systems introduce P&A features into HLBSs [Boyle and Encarnacion, 1994, Kobsa et al., 1994, Brusilovsky et al., 1996a, Vassileva, 1996, Höök et al., 1996], much of this research has taken a hands-on, technology-driven approach.

The aim of PA-HLBSs research is to increase the functionality of HLBSs by making the user interaction process personalisable. The approach taken is to enhance HLBSs with P&A features which may be initiated by the users of such systems, or by the system itself.

Such systems are assumed to be useful in any application area where users have different information seeking goals, histories and preferences. PA-HLBSs aim to address this issue by using knowledge about the user to tailor the information and links presented on an individualised basis. Furthermore, it is argued that P&A features can assist the users when interacting with large hyper-networks [Brusilovsky, 1996a]. By applying knowledge of users, such systems can support users in their navigation by limiting the options for traversal to information units, suggesting relevant links to follow and providing additional information on links and information units.

Several PA-HLBSs have been developed for this purpose including Adaptive Hyper-Man [Myka et al., 1992, Mathe and Chen, 1996], ELM-ART [Brusilovsky et al., 1996a], Hynecosum [Vassileva, 1996], PUSH [Höök et al., 1996] and KN-AHS [Kobsa et al., 1994].

This dissertation describes an abstract approach to the characterisation of hyperlink-based interaction. This approach is intended to precisely identify and formalise the unique properties of HLBSs as opposed to those issues relating to technologies of which HLBSs tend to be a client (e.g., user-interface, database and link management technologies).

Although this aim is shared with OHLBSs research, the focus of this dissertation falls on inducing from that characterisation a rich set of P&A actions, rather than on an investigation of the interactions which take place between HLBSs and server technologies. The characterisation then forms a model which can be used as a gateway to a principled exploration of specific P&A issues within HLBSs. The characterisation also lends weight to the argument for the delegation to specific research areas (e.g., database and user-interface research) the task of exploring P&A issues in their own specific contexts.

The remainder of this introduction comprises three parts:

1. **Motivation and Challenges**: In the first part, the motivation and the challenges faced in carrying out the research are reported;

2. **Technical Aims of Research**: In the second part, the technical development of the research and the central questions addressed are introduced;

3. **Contribution**: Finally, in the third part, the research results contributed are summarised.

Together, these three parts of the introduction aim to convey a sense of where the main contributions of the dissertation lie and how they articulate and cohere.

### 1.2 Motivation and Challenges

This section presents an analysis of the motivation and challenges faced by PA-HLBSs researchers in general and more specifically those which underpin the research reported herein.

#### 1.2.1 Motivation

Research into P&A actions in HLBSs is motivated by the need (both scientific and commercial) to increase the effectiveness of HLBSs as a platform for information retrieval.
tasks when users have different information goals and histories.

Most HLBSs implicitly assume that users:

1. have similar information goals;
2. have similar backgrounds;
3. undertake user interaction along similar lines.

These implicit assumptions have resulted in many such systems not allowing users to tailor the information they contain or the user interaction process in significant ways. One consequence of this state of affairs is the establishment of conservative bounds on the efficiency and effectiveness of the current generation of HLBSs.

This analysis suggests that there might be gains if it were possible to move to a more personal, responsive approach to user interaction with HLBSs. Figure 1.1 depicts the classic motivation for personalisation and adaptation in HLBSs.

![Motivation for Personalisation and Adaptation in HLBS](image)

Figure 1.1: Motivation for Personalisation and Adaptation in HLBSs

It is now generally accepted [Brusilovsky, 1996b] that users of HLBSs may differ in their information goals insofar as they may have preferences as to what information is provided, which links are used to navigate the information space and where the information units to be obtained may be sourced. Users may also differ in their histories insofar as they are likely to have different knowledge of the information contained within the HLBS, of the information space and how it may be navigated, or of the operation of the system itself. Not only are these differences evident among different users at any given point in time, they also manifest themselves for the same user at different points in time (e.g., because the user becomes more or less knowledgeable).
These observations suggest that the effectiveness of HLBSs could increase significantly if users could personalise their interaction with HLBSs (and, more ambitiously, have the system adapt interaction to better suit individual users). In this dissertation, this suggestion is referred to as the benefit-gain hypothesis.

Most of the interaction a user might experience with a hyperdocument is determined by the design decisions that shaped the hyperdocument in terms of its content, rendering and navigation possibilities (i.e., its links). Typically, the designers decide the content that is to be rendered, the rendering attributes that determine the appearance the content will take and the links that determine which other hyperdocuments the user can navigate to.

As a consequence of the fact that these decisions are unilateral and irreversible (i.e., cannot be overridden by users as they interact with the hyperdocument), it can be said that the designers own the hyperdocument. Thus, in HLBSs that lack P&A features, users can navigate through a hyperdocument using links, but they are prevented from enforcing their individual preferences as to content, rendering and navigation possibilities.

The research in this dissertation is motivated by the view that merely providing users with browsing facilities does not realise the potential effectiveness of HLBSs in many important information retrieval tasks. This research process proposes one approach to overcoming this impediment by extending HLBSs with formally defined P&A actions that effect a transfer of ownership from designers of the hyperdocument to each of its users, thereby enabling the latter to redesign the former according to their specific information goals and histories.

Motivation is also drawn from the fact that there seems to be no discernible consensus among researchers with respect to the following fundamental questions:

1. Which are the emergent properties of HLBSs? Equivalently, what is the scope of P&A in HLBSs?

2. Which P&A actions could be made available to users? Equivalently, what descriptive stance should be taken with respect to P&A actions in HLBSs?

3. Which P&A actions should be made available to users? Equivalently, what prescriptive stance should be taken with respect to P&A actions in HLBSs?

In this dissertation it is argued that a precise, abstract characterisation of what emergent properties can be assigned to HLBSs is required, so that issues relating to P&A in the technologies, of which OHLBSs tends to be a client, are not compounded with P&A issues in HLBSs. Such a characterisation would open the way to a principled exploration of specific P&A issues in HLBSs and would also justify delegating to specific research areas.

---

4In this dissertation a rendering unit in a HLBS is referred to as a *hyperpage*, and a collection thereof as a *hyperdocument*. Later sections make these notions more precise in the context of this dissertation.
(for example, database and user-interface research) the task of exploring P&A issues in their own specific contexts.

The scope for P&A actions in HLBSs, it is argued in this dissertation, comprises exactly the emergent properties of HLBSs when these are viewed as only loosely coupled to a variety of servers, in each of which the scope for orthogonal P&A actions can also be characterised. It is the specificity of HLBSs, and hence the scope for P&A actions they might support, that the research reported in this dissertation seeks to identify at an abstract level above that of concrete systems.

A conceptualisation that addresses this issue is fundamental not only to an understanding of what HLBSs are, but also to how their essential characteristics might be exploited to increase their effectiveness in information retrieval tasks such as certain aspects of learning.

The lack of large-scale empirical studies into the effectiveness and efficiency of P&A actions that could back up a prescriptive stance suggests that a descriptive stance is more advisable at this stage in the research enterprise. As there is no prior reason to constrain the space of possibilities, one is led to conclude that, in principle, all the decisions that designers make when composing hyperdocuments are potentially within the scope of P&A actions. As far as the author is aware this expressiveness has not been modelled and is not offered by any of the PA-HLBSs.

The most likely explanation for this state of affairs is the lack of an abstract model of HLBSs that characterises precisely their emergent properties and thereby makes possible a systematic, principled and exhaustive elicitation of the space of design decisions (equivalently, the space of possible P&A actions). It is only by drawing this space of possibilities that future research will be able to test, in a principled manner, the hypothesis that HLBSs that provide P&A features are more effective in information retrieval tasks than those that do not provide such features.

A further motivating factor for research into PA-HLBSs is the need to address the classical HLBSs issues of cognitive overload and disorientation, as discussed in [Conklin, 1987] and [Thüring et al., 1995]. It is proposed that transferring ownership of a hyperdocument from designer to users allows for alterations to the content, rendering and navigational properties of a hyperdocument that minimise the levels of cognitive overload and disorientation suffered in navigating the information space made available by the HLBS. For example, users may use P&A actions to reduce the number of links displayed or to annotate links with visual clues as to the result of traversing a link. Furthermore, users may use P&A actions to restrict the content available to a subset which they deem to be more appropriate to their own information goals and histories.

However, it is recognised that the introduction of P&A features into a HLBS may add an extra layer of complexity which could have an effect on the levels of cognitive overload and disorientation suffered.
1.2.2 Challenges

Although there are many existing models of a variety of HLBSs and issues, there is still no model which facilitates a systematic investigation of the space of possibilities for P&A actions. The hands-on, technology-driven ad hoc approach that has prevailed makes a principled testing of the benefit-gains hypothesis difficult. In many cases, it is far from clear whether what is being tailored is a unique, distinctive, identity-defining property of hyperlink-based interaction or, instead, an emergent property, such as one that simply arises from the tight coupling of database and user-interface technology.

This is an important conceptual distinction insofar as the latter case simply makes use of the functionality that database and user-interface technology provide as a matter of course to all clients, including HLBSs.

The first challenge which this research process aims to meet is, therefore, a characterisation of a formal, conceptual model of core hyperlink-based functionality which assumes, but is not determined by, data or user-interface services. Such a characterisation may be used to distinguish between essential properties and emergent properties of HLBSs.

The claim that making HLBSs personalisable and adaptive increases their efficiency and effectiveness as information retrieval systems, at present, suffers from a lack of empirical evidence. For PA-HLBSs researchers to provide this empirical evidence, the following questions should be addressed:

1. Why are P&A actions required?
2. What is the purpose of P&A actions?
3. Which P&A actions could be made available to users?
4. Which P&A actions should be made available to users?

The response so far from the body of research into PA-HLBSs has been to build P&A features to allow users to control the interaction process which takes place between themselves and HLBSs.

However, P&A features that have been made available in PA-HLBSs are often those which are feasible to implement in some system, with whatever technologies are currently available and which the intuition of their researchers suggested would be useful. Furthermore, at present there seem to be no clear answers to the other questions, relating to which P&A actions could or should be made available.

The second challenge which this research process aims to address is, therefore, how to model at a suitable level of abstraction the space of possibilities for P&A actions that could be made available to users of PA-HLBSs. Such a model should characterise the notion
of a “transfer of ownership” and should avoid being technology-driven. Furthermore, the choice of P&A actions should fall out from this abstract model of personalisable, adaptive hyperlink-based interaction and should ultimately be subject to empirical tests for effectiveness gains.

It is envisaged that by modelling the space of possibilities for P&A actions, research can be conducted which will allow the testing, in a principled manner, of the hypothesis that HLBSs that provide P&A features are more effective in information retrieval tasks than those that do not. Furthermore, such research may highlight those P&A features that are more effective than others.

1.3 Technical Aims of Research

The technical development underlying the solution to the characterisation of PA-HLBSs is now briefly described. First, the central questions addressed are stated, then their derived problems are outlined and the approach taken to solve these problems detailed. Finally, the methods used to achieve a solution are presented.

1.3.1 The Central Question

The central questions which this dissertation aims to answer are; what is the scope of P&A in HLBSs and which P&A actions could be made available to users?

In presenting a solution, a formal approach is taken that seeks to build upon as much of the theory and practice of HLBSs research to date as possible. The ultimate goal is to contribute an abstract formal characterisation of personalisable, adaptive hyperlink-based interaction.

1.3.2 Derived Problems

In answering the central questions, the following derived problems present themselves:

1. How to model interaction with a hyperdocument?
2. How to transfer ownership of the process of interaction with a hyperdocument from the designers to its users?
3. How to make design decisions more explicit?
4. How to make design decisions revisable?
5. How to distinguish, model and implement non-tailorable and tailorable hyperdocuments?
6. How to distinguish, model and implement personalisation and adaptivity?

An analysis of existing responses to these problems elicits the following shortcomings:

- It can be argued that the predominant approach has been to choose a tailoring action and then implement it using a new or existing system. Such an approach means that it is often difficult to generalise and gain insights into any underlying principles;
- The motivation and design decisions made in the process of implementing tailoring actions are often hard to phrase and therefore revise;
- Often the notions of personalisation and adaptivity are not clearly distinguishable, resulting in some degree of conceptual muddle;
- It is difficult to analyse and measure the presumably distinct, independent effects of tailoring actions.

The research reported in this dissertation aims to respond to these shortcomings via the following goals:

1. To characterise a core of hyperlink-based functionality viewed as a client technology;
2. To view tailoring actions as ranging over entities within this core and as effecting the transfer of ownership from designers of HLBSs to their users;
3. To make design decisions explicit and therefore revisable, through the use of formal modelling techniques;
4. To model personalisation as user-initiated tailoring of hyperdocuments and adaptivity as system-initiated tailoring;
5. To propose this distinction as a basis to analyse and measure their independent and compound effects.

In achieving these goals this research process aims to provide an account of what PA-HLBSs researchers perceive the important characteristics of personalisable hyperlink-based interaction to be.

A further goal is to incorporate the following principles for personalisation, which emerged from preliminary research:

1. Personalisation should, to some degree, represent the measured transfer of ownership of the process of interaction with a hyperdocument from designer to user;
2. All hyperpage design decisions should, in principle, be able to be the subject of personalisation requests;

3. A model of hyperlink-based personalisation should aim to accommodate all recognised personalisation actions (see [Brusilovsky, 1996b] for a comprehensive review);

4. Personalisation actions should be clearly defined, explicit and capable of being formally defined;

5. All personalisation actions should be consistent, repeatable and revisable;

6. Any choice of personalisation actions should ultimately be subject to empirical tests for effectiveness gains.

This methodological procedure of isolating the scope for P&A, defining it formally and then inducing from that formalisation the set of P&A actions may also be seen as a contribution that can be used in other settings, under different assumptions and using alternative conceptualisations of HLBSs.

1.4 Research Contribution

The following section details the main contributions made by this dissertation, the major contribution being a comprehensive formalisation of personalisable, adaptive hyperlink-based interaction. The particular attraction of such an approach is that P&A actions can be studied with greater conceptual clarity than is possible by technology-driven experimentation. Furthermore, the formalisation proposed is cast at a level of abstraction above that of concrete systems exploring current technologies, thereby allowing for a principled exploration of specific P&A issues. The contribution made by this research is unique, as far as the author is aware, in formally modelling a rich set of abstract user-initiated tailoring actions, which enable users to come closer to satisfying their specific, often dynamic, information retrieval goals. It indicates how system-initiated tailoring actions (adaptivity) fit cohesively and non-disruptively with user-initiated ones (personalisation).

In this dissertation the following distinction is drawn between personalisation and adaptation. Any action that effects, or helps effecting, transfer of ownership (in the broad sense discussed above) is referred to as a tailoring action. Personalisation actions are user-initiated tailoring actions. Adaptive actions are system-initiated personalisation actions. Consequently, personalisation (respectively, personalisability) is the process (respectively, the capability) of carrying out personalisation actions, and adaptation (respectively, adaptivity) the process (respectively, the capability) of carrying out adaptive actions.
In this dissertation attention is drawn to the fact that system-initiated tailoring is, in principle, as expressive as user-initiated tailoring and requires no other technologies than those involved in user modelling and in decision making from a user model. That is not to say that those are not extremely challenging problems, rather that, in the model proposed in this dissertation no additional concern is needed.

However, to implement concrete systems which enable system-initiated personalisation, a user model and decision making algorithms will be required. In this dissertation an architecture and decision making algorithm is proposed that depicts an approach to add, in a non-disruptive manner, adaptive capabilities to the proposed model. The approach centres on an adaptation function driven, in part, by a set of active rules. This function implements an inference engine over a decision theory (i.e., a theory as to which actions are more likely to yield the most benefits given some accumulated knowledge of past interactions). The accumulated knowledge is the information goals and the history of each user, while the actions which the inference engine is in charge of suggesting are P&A actions as defined for user-initiated tailoring.

1.4.1 A Formal Model

This research contributes an abstract model of personalisable, adaptive hyperlink-based interaction. The main contributions are, first, the definition of an abstract model of core hyperlink-based functionality, and second, the definition of an abstract model of personalisability extending the core model. Furthermore, an architecture is drawn which indicates how adaptivity fits into this abstract model and what broad lines its realisation would take.

The core is formalised as a composer from specifications, i.e., what the designer of a hyperpage designs is not a hyperpage but rather a specification of how to build the hyperpage upon request. Hyperpages are modelled as formal specifications and a formal language has been defined for this purpose. Hyperpage specifications are restricted to determining where to source content and how to render it.

Defining hyperpages to be specifications allows the proposal of a model in which they may be generated dynamically, i.e., the renderable text of a page $P$ is composed on the fly and may incorporate responses to requests that might in this case have been addressed to various servers when the user chose to navigate into $P$. The model is cast at a level of abstraction which enables any data querying specification language (e.g., SQL [Abiteboul et al., 1995]) and mark-up specification language (e.g., HTML [Raggett et al., 1999]) to be used in the specification of hyperpages.

The semantics of hyperpage specifications are given with reference to a formal abstract machine. The abstract machine illustrates the execution of hyperpage specifications, thereby yielding renderable hyperpages.
Induced from the formal definition of hyperpage specifications is a set of annotation possibilities and P&A actions. These enable all design decisions realised as hyperpage specifications (and recursively their component parts) to be revised.

Personalisation is modelled as the user-initiated process of annotating and rewriting a hyperpage specification into a version thereof that is associated with the user who took that action. It follows that the hyperpages that users see rendered may, if they wish, reflect their preferences, shaped by their information goals and histories.

Personalisation is viewed as the process of handing over to the user the ability to annotate specifications or to rewrite them or both, thereby allowing the user to override, in principle, each and every designer specification.

When personalisation functionality is layered over the core a designer can annotate a hyperpage in preparation for differences in users’ goals and histories. A user can request to personalise not only such annotations, but the hyperpage specifications as well. Personalisation requests allow users to specify which hyperpages are to be personalised and how they should be transformed. Therefore, they effect a transfer of ownership of all design decisions.

Hyperpage annotations and personalisation requests are modelled as formal specifications and formal languages have been defined for this purpose. Set theoretic and relational algebraic expressions are used to represent the semantics of personalisation requests. For this purpose the relational model [Codd, 1970] is used to represent hyperpage and annotation specifications. A parser for the source language of hyperpages, annotations and personalisation requests has been specified and implemented in Prolog [Clocksin and Mellish, 1994].

Adaptivity is viewed as the process of allowing the system to take the initiative in personalisation actions, in the light of the system’s inference of a user’s information goals and history. When adaptivity functionality is layered over the core and personalisation layers, the following capabilities are added to the proposed model. Both users and designers can define strategies as to when the system should take the initiative and actively tailor the interaction to a user in the light of that user’s information goals and history of use. Adaptation is viewed as system-initiated tailoring. Broadly, when the HLBS identifies an opportunity to adapt, it does so by choosing to carry out an action which the target user might have carried out were he or she motivated to do so.

To summarise, the model presented in this dissertation contributes the following answers to the basic P&A questions proposed in 1.2.1:

1. Why are P&A actions required?
   
   P&A actions are required to test the efficiency and effectiveness of HLBSs in certain contexts;
2. What is the scope of P&A (both user and system-initiated) in HLBSs?

A view of a core of hyperlink-based functionality is adopted and formalised. This view presupposes a variety of servers to which HLBSs are only loosely coupled. It is shown how this functionality emerges from a formal definition of hyperpages. It is then proposed that it is the formal elements in this formal definition that are within the scope of P&A actions. When such formal elements are personalised the composition of specifications into renderable documents effects the transfer of ownership;

3. Which P&A actions could be made available to users?

It is proposed that all formal elements in the formal definition of a hyperpage are within the scope of P&A actions. This view essentially means that every decision made by the designers can be overridden by a P&A action. Equivalently, every hyperpage can be entirely redesigned by the set of P&A actions induced from the proposed abstract model. In this dissertation it is argued that empirical studies should be carried out before any prescriptive stance is adopted.

In this research it is felt that the remaining question, addressing which P&A actions should be made available to users, requires empirical studies conditional on the answers proposed by the research reported here. These empirical studies, as a result, lie outside the scope of the research enterprise reported in this dissertation.

The main ideas guiding the approach taken to modelling personalisable hyperlink-based interaction can therefore be phrased as follows:

1. The model is an abstract model, as many steps removed from concrete implementations as necessary to allow a systematic, exhaustive investigation of P&A issues in HLBSs;
2. The model is an open model, insofar as HLBSs are viewed as clients of a variety of servers and in particular of data and user-interface servers;
3. Personalisation involves a transfer of ownership of hyperdocuments, from designers to users;
4. To ensure that the set of P&A actions are consistent and exhaustive, its elements are induced from the formal definition of the hyperdocuments they act upon;
5. All design decisions are, in principle, within the scope of personalisation actions;
6. The model of hyperlink-based personalisation can express most personalisation actions proposed in the literature (see [Brusilovsky, 1996b] for a comprehensive review);
7. The model describes which personalisation actions could be made available.
1.4.2 PAS: A Personalisable Hyperlink-Based System

This dissertation reports on the development of a personalisable HLBS referred to as PAS. The aim of PAS is to instantiate the major aspects of the proposed model. The main contribution of PAS is to provide some evidence in support of the claim that the abstract approach taken allows not only for a greater understanding of what personalisation and adaptivity means in the context of HLBSs, but also how the model may aid the design of such systems.

The PAS system consists of two major components: an instantiation of a WWW-based application for realising dynamically generated hyperdocuments; and an instantiation of a parser for hyperpages and personalisation requests. The PAS system has been implemented using client-server technologies for WWW-based data retrieval and presentation.

The PAS system will be used in future work as a foundation for a systematic investigation of the effects of P&A actions.

1.5 Thesis Structure

This introduction has discussed the background, motivation and challenges for the research reported in this dissertation. The central question which the research endeavours to solve has been stated together with the methods through which a solution was achieved. Finally, the contributions of this dissertation were briefly sketched. The remainder of this dissertation is structured as follows:

- **Chapter 2** is dedicated to PA-HLBSs research issues. The chapter aims to place the research reported in this dissertation in context by summarising the evolution of the major trends and issues related to the modelling, design and implementation of HLBSs;

- **Chapter 3** presents a detailed discussion of HLBSs and PA-HLBSs modelling issues. This chapter introduces a formal model for personalisable, adaptive hyperlink-based interaction. This chapter also places in context the formal model for PA-HLBSs, which is the major contribution of the research reported in this dissertation;

- **Chapter 4** proposes a model of non-personalisable, non-adaptive hyperlink-based interaction modelled by a group of functions referred to collectively as the \textit{H-region}. The most salient feature of the H-region is a formally defined core of hyperlink-based functionality which is separate from the database management and user-interface components which HLBSs necessarily include. The model presented is used to define what, within HLBSs, can be subject to personalisation initiatives. In
this chapter a formal language for hyperpage specifications is presented and their semantics formalised;

- **Chapter 5** proposes a model of user-initiated, personalisable hyperlink-based interaction. The model overlays the component that described in chapter 4 with a further group of functions, collectively referred to as the *P-region*, so that user-initiated tailoring actions are made possible. The P-region comprises a group of functions that are non-disruptively added to the H-region in order to model personalisable hyperlink-based interaction. When the P-region is added users can not only request a hyperpage, but also annotate or rewrite it, thereby creating their own version of it. The decisions that the designers of that hyperpage have made with regard to content, navigation and rendering of the hyperpage can therefore be overridden by the user. This causes no disruption whatsoever and requires no changes at all to the H-region. In this chapter both the formal language specifications for hyperpage annotations and personalisation requests are detailed and the semantics of personalisation requests formalised;

- **Chapter 6** proposes further extensions to the model detailed in chapters 4 and 5 to allow for system-initiated, personalisable hyperlink-based interaction. The model overlays that described in chapter 5 with a further group of functions, collectively referred to as the *A-region*. The A-region comprises a group of functions that are non-disruptively added to the P-region to model adaptivity. Adaptation is viewed as system-initiated tailoring. The A-region enables users and designers to define strategies as to when the system should take the initiative and actively tailor the interaction with a user in the light of that user’s information goals and history;

- **Chapter 7** provides some evidence in support of the claim as to the practicality of the formal approach taken with a discussion of how it has been used in the development of PAS, a prototype implementation of a PA-HLBSs developed by the author;

- **Chapter 8** compares and contrasts the contributions and limitations of the proposed model with that of related work into HLBSs and PA-HLBSs;

- **Chapter 9** closes the dissertation by summarising the research contributions made and future directions resulting from the approach taken to model PA-HLBSs. Possible application domains for the work reported in this dissertation are also discussed. The chapter closes with a summary of research conclusions.
Chapter 2
Personalisable, Adaptive HLBSs
Research Issues

To place this dissertation in its proper context this chapter provides a detailed analysis of the main lines along which research into HLBSs and, in particular, PA-HLBSs has developed. It is hoped that this analysis will highlight the main motivations of the research as well as the work that is, in varying degrees, related to it.

The chapter opens with a discussion of the cognitive basis of hypertext, hypermedia and more generally HLBSs. The purpose of this discussion is not only to give the reader a sense of what HLBSs are, but also to indicate the need for a more general definition which highlights their uniqueness as models of information retrieval, namely the linking of units of information in an associative manner.

Following this discussion the historical context in which PA-HLBSs have evolved is analysed. It is hoped that this analysis goes some way towards illustrating how HLBSs and PA-HLBSs have been realised so far and may be realised in the future.

2.1 Understanding Hyperlink-Based Systems

2.1.1 The Cognitive Basis of Hyperlink-Based Systems

In order to understand HLBSs research it is essential to first consider how users seem to read and write documents. Human intelligence depends upon models with which people communicate. Broadly, it is believed that one associates pieces of information with other information to create complex knowledge structures or networks of information. Such a network forms a mental model which may be viewed as a semantic network comprised of units of information represented as nodes, and associations between units represented as links [Rada and Wang, 1995]. A communicator must be able to persuade the audience that acceptance of their communicated mental model is in the best interests of the audience.

During this process the typical medium (i.e., paper, video) requires a sequential presentation. An author transforms their non-linear communication (mental model) into
a linear form. The author is said to have to “linearise” their mental model to prepare it for communication. As the linear representation is not natural, authors may provide additional information (i.e., tables of contents and indexes). Readers, in turn, transform the linear message into their own non-linear models in their minds. Generally this process involves the reader breaking up the information into smaller parts and then rearranging these parts based on their information goals. This process is shown in figure 2.1.

![Diagram of linear message](image1)

**Figure 2.1:** Process of Reading and Writing using Traditional Linear Media

![Diagram of non-linear message](image2)

**Figure 2.2:** Process of Reading and Writing using a HLBS

At ACM hypertext’87 a writing model was suggested which consisted of three cognitive phases [Smith et al., 1987]:

1. *exploring phase* during which knowledge is formulated, initial drafts are made, and ideas are grouped into different perspectives;

2. *organising phase* during which initial drafts are organised into sequence and an outline is made;
3. *encoding phase* during which the document is written.

Although these phases may be undertaken in sequence, generally authors like to move freely from one phase to another. In [Delisle and Schwartz, 1989], the following activities are described:

1. *Recording relevant ideas* where initial drafts of ideas are made;
2. *Exploring relationships among ideas* which is the associative linking process;
3. *Structuring ideas* which is the process of organising drafts and creating an outline;
4. *Document preparation* during which the document is written, linearised, formatted and printed.

Research into HLBSs is motivated by their ability to enable authors to partially mimic the processes of writing described above. The logical model of HLBSs is largely that of a semantic network based on the associative linking of units of information attached to the nodes of the network. The analogy of a semantic network to a hyper-network is straightforward and has long been recognised [Conklin, 1987, Rada, 1990]. HLBSs allow authors to directly create networks that communicate their mental models through the use of computer supported links. Authors may not only recreate their mental models as non-linear information structures (hyper-networks) but also avoid the process of linearising the information they wish to communicate. This process is shown in figure 2.2.

In summary, HLBSs allow users to access units of information through associations made between these units. Furthermore, information construction through association seems to be the natural way in which the human intellect works.

### 2.1.2 Defining Hyperlink-Based Systems

In 1965 Nelson coined the term *Hypertext*, defining it as:

> a body of written or pictorial material interconnected in such a complex way that it could not conveniently be presented or represented on paper [Nelson, 1965].

Two interesting observations can be drawn from the above quotation. Firstly, it emphasises the structure of hypertext (material interconnected) to be its defining aspect and secondly, no differentiation is made between the interconnection of various forms of media. Nelson believed that “everything is deeply intertwingled” and used the word ‘hyper’ to mean extending a document by connecting it to others. From its origins therefore, hypertext
was envisaged to be hypermedia (text and other media) with a primary concern of inter-
connecting material in a convenient manner.

Since the introduction of the term ‘hypertext’ many researchers, with varying degrees of success, have attempted to provide more precise definitions of both hypertext and hypermedia [Shneiderman and Kearsley, 1989, Newcomb et al., 1991, Lowe and Hall, 1999]. What can be concluded from an analysis of these definitions is that the distinctive characteristic which they seek to capture is that of connecting units of information together to facilitate access to that information in a manner which is closer to the way users naturally interact with information (i.e., associative linking). This point is best illustrated by the following three different interpretations of hypertext and hypermedia:

Hypertext: a database that has active cross-references and allows the reader to ‘jump’ to other parts of the database as required. [Shneiderman and Kearsley, 1989];

A hypertext document is a parcel of written and graphic information intended for human perception, which can be explored and presented in a variety of sequences using a set of traversable connections usually called ‘links’. [Newcomb et al., 1991];

An application which uses associative relationships among information contained within multiple media data for the purpose of facilitating access to, and manipulation of, the information encapsulated by the data. [Lowe and Hall, 1999].

It is clear that although these interpretations differ\(^1\) in their approach, the underlying theme is one of using links between information units (i.e., cross-references, traversable connections, associative relationships) to access information. Furthermore, the now generally accepted view that such systems may, and usually do, include the possibility for working with graphics and various other media leads one to conclude that it may be more appropriate not to differentiate between hypertext and hypermedia, but rather to promote their distinctive characteristic, namely linking. To quote Bush, “The process of tying two items together is the important thing” [Bush, 1945]. Therefore, throughout this dissertation the term hyperlink-based system is used in preference to hypertext and hypermedia not only because it is more general, but also because it highlights this truly distinctive characteristic.

\(^1\)The first definition may be viewed as data-oriented, the second, product-oriented and the third, goal-oriented.
2.2 Historical Context

The historical context in which HLBSs have evolved can be described as both rich and influential in terms of how users interact with and use computers today. Rich in the sense that HLBSs research is associated to some degree with the vast majority of research into computer-based user interaction. Influential in the sense that with the mainstream emergence of large-scale HLBSs such as the WWW, HLBSs interaction is emerging as the predominant mode of computer-based information retrieval.

2.2.1 Origins of HLBSs Research

The first speculative discussion of the concepts underlying HLBSs can be found in Bush’s proposal in 1945 for a system called *memex* (“Memory Expander”) [Bush, 1945]. Bush described the memex as, “a sort of mechanised private file and library”. The memex system would store an individual’s information on microfilm, which would be kept in the user’s desk. This desk was intended to have several microfilm projection positions to enable the user to compare different microfilms, in a manner very similar to the windows systems that have become popular on personal computers in the last twenty years.

After stating the mechanics of memex, Bush stated that,

All this is conventional, except for the projection forward of present-day mechanisms and gadgetry. It affords an immediate step, however, to associative indexing, the basic idea of which is a provision whereby any item may be caused at will to select immediately and automatically another. This is the essential feature of memex. The process of tying two items together is the important thing. [Bush, 1945]

In the quotation above lies a central goal of HLBSs research, namely to construct models and systems which exhibit forms of associative indexing. Associative indexing may be viewed as the instantiation of a semantic relationship, or “link” between units of information. Furthermore, Bush argues that,

Our ineptitude in getting at the record is largely caused by the artificiality of systems of indexing... The human mind does not work that way. It operates by association. [Bush, 1945]

This quotation highlights the importance Bush placed on the user in the process of constructing associative relationships between units of information. What may be inferred is that the semantic network constructed as a result of associative indexing should, to some
degree, mimic the complex knowledge structures (semantic networks) users seem to build in their own minds when trying to make sense of related information.

An important conclusion that can be drawn is that the memex system was envisaged to be personalisable, in that a user could make an associative relationship or link between two units of information. It can be observed that even today the vast majority of HLBSs provide each user with the same associative relationships (retrieval possibilities) disregarding users’ specific information goals and histories. Furthermore, all decisions as to the association of information units are decided by the designer of the system, with users given no opportunity to interfere with these decisions. To summarise, most HLBSs today are not personalisable by the user.

The early 1960s saw two researchers, Nelson and Engelbart, begin work separately on what have become the foundations of HLBSs research. Nelson worked and is still working on a system called Xanadu [Nelson, 1967, Nelson, 1980, Nelson, 1982]. The focus of this research is to extend Bush’s original ideas of managing and retrieving information to using what he termed ‘hypertext’ to address ways in which information could be used and manipulated.

The basic goal of the Xanadu system is to act as a repository for all the information anybody had ever written. Xanadu is to store more than just documents and the links between them, but also all versions of all documents. Users would not be able to delete documents, even when they had been updated, as others may have already made links to previous versions of a document. Thus the Xanadu system is viewed as a truly universal “hypertext” system.

The crucial implication of the Xanadu system is to view HLBSs as being concerned not only with information retrieval, but also information usage. The Xanadu system, and Nelson’s work, in general seeks to address issues such as intelligent browsing of information and user navigation. One conclusion that can be drawn in the context of this dissertation is that Nelson saw information retrieval, even when associative relationships between documents had been made, to be a difficult task and saw that users would require assistance. A second conclusion is that HLBSs should be active in the sense that they should assist not only in navigation, but also provide feedback and even mechanisms for active annotation on an individualised basis. Through an examination of HLBSs deployed to date it is fair to conclude that the vast majority of such systems have yet to achieve the functionality described in Nelson’s work.

In 1962 Engelbart began working on the Augment project. This project was concerned with developing tools to augment human capabilities and productivity. A major aspect of the Augment project was NLS (oN-Line System) [Engelbart et al., 1973, Engelbart, 1995] which, although not strictly a HLBS, had several hyperlink-based features. NLS allowed links to be made between research papers stored in a central “journal” facility.
Significantly, in 1968 Engelbart provided the first public demonstration of many of the basic ideas of hyperlink-based interaction when he demonstrated NLS.

The first operational HLBS, known as the *Hypertext Editing System*, appeared in 1967 [van Dam, 1988]. The system developed at Brown University in the USA was a mainframe based HLBS.

In 1978 the first hypermedia system, the *Aspen Movie Map* [Nielsen, 1995] developed at the Massachusetts Institute of Technology was demonstrated. The town of Aspen, in Colorado (USA) was filmed by driving a truck through every street and taking front, rear, left and right view photographs every three meters. These were transferred to videodisc and linked to follow the street grid. Users sat in front of a vertical monitor showing the street view and a flat monitor showing an overview map and could navigate through the streets of Aspen via a joystick. Short video clips of many of the buildings in Aspen were also hyperlinked.

During the early 1980s ever more sophisticated HLBSs were developed. Significant examples include *Guide* [Brown, 1987] , *Intermedia* [Yankelovich et al., 1988, Riley, 1990] and *HyperCard* [Goodman, 1987].

*Guide* [Brown, 1987] was the first widely available commercial hyperlink-based system, originally developed at the University of Kent in the UK and released as a commercial product by OWL [Hershey, 1987] in 1986 for the Macintosh computer and later for the IBM compatible PC market. This system marked the transition of HLBSs from the research community to real-world applications.

*Intermedia* [Yankelovich et al., 1988, Riley, 1990] developed at Brown University from 1985 to 1990, introduced the concept of link anchors and the use of a separate database for links (rather than storing them within documents), allowing links to be bidirectional and hyper-networks to be maintained as distinct entities. Intermedia was a multi-user system based on the client-server architecture and combined hyperlink-based features with information retrieval facilities.

HLBSs were further popularised by Apple Computer’s *HyperCard* [Goodman, 1987] given free with every Macintosh computer from 1987. HyperCard is a frame-based, stand-alone HLBS incorporating a simple, but powerful, scripting language called HyperTalk.

Since its emergence in the 1990s, the WWW [Berners-Lee, 1991, Berners-Lee et al., 1992] has become the most popular platform for the deployment of HLBSs. The WWW is now by far the most popular HLBS. A further breakthrough occurred in 1993 when the National Center for Supercomputing Applications (NCSA) released Mosaic, a graphical browser for the WWW [Duval and Main, 1994].

The unabated growth of Internet sites and the popularisation of web pages has led to the WWW permeating most walks of life. However, many researchers have criticised
the WWW, not only for the quality of information available, but also for the focus that is placed on information provision rather than information usage [Lowe and Hall, 1999, Nielsen, 1995]. In general, the WWW has been criticised for its lack of active support for user navigation and information searching. Although the WWW is now supported by many search engines [Gudivada et al., 1997, Lawrence and Giles, 1998], the vast majority of these merely provide keyword and phrase-based searching. In part, this situation has arisen as a result of the fact that the hypertext mark-up language (HTML) [Raggett et al., 1999] used to design the vast majority of web pages makes little provision for web page annotation. This issue is a strong motivating factor for the research reported herein. In this dissertation a model of hyperdocuments (of which a web page is one instantiation) is proposed which allows for extensive annotation of a hyperdocument and its component parts.

Furthermore, HTML does not provide data querying facilities to enable designers of web pages to specify that content for a web page may be served from a server, perhaps remotely. Rather it relies on the designer entering the content directly into the web page specification. The abstract model reported in this dissertation seeks to address these limitations by modelling hyperdocuments as specifications which determine the source of content and how to render it. Recent proposals on XML by WC3 [Bray et al., 1998] and commercial products such as Cold Fusion [Danesh and Kristin, 1999, Forta, 1999] and Microsoft’s Active server pages [Chappell and Linthicum, 1997, Ladd, 1997] illustrate the acceptance of the limitations of HTML and indicates the surge of interest in new technologies which allow database-originated content to be incorporated within web pages.

In summary, the WWW has evolved to become the most widely used HLBS to date. It has successfully allowed the mass publication of hyperlink-based information. However, its limitations in addressing issues of user navigation and database access have been widely acknowledged and are currently being addressed.

2.2.2 Models of HLBSs

By the late 1980s, intense hands-on development activity resulted in a number of different HLBSs being implemented. However, a need for a greater understanding of issues of structure, content and navigation in HLBSs [Bernstein et al., 1991, Botafogo and Shneiderman, 1991] led to a surge in interest on generalising these instances into abstract models of HLBSs.

At the first ACM conference on hypertext in 1987, Campbell and Goodman proposed the notion of a Hypertext Abstract Machine (HAM) [Campbell and Goodman, 1987]. The HAM is a milestone in the development of HLBSs modelling as it is the first major attempt to define an abstract model in which HLBSs could be expressed. Then, in two workshops [Moline et al., 1990], the first of which was held in October 1988, a group of
leading HLBSs researchers collaborated to propose a formal model of HLBSs, known as the *Dexter model* [Halasz and Schwartz, 1990, Halasz and Schwartz, 1994], aimed at fixing a terminology and providing a formal specification that generalised over most implemented HLBSs then in existence.

Other models for HLBSs which emerged from the same workshops included the Trellis Hypertext Reference Model [Furuta and Stotts, 1990], the Strawman Reference Model [Thompson, 1990] and the Formal Model [Lange, 1990].

Despite the variety of models proposed in [Moline et al., 1990], the following shortcomings suggest that these early models may no longer be appropriate to explain the latest generation of HLBSs.

Although these models characterise how HLBSs manifest themselves, they fail to address the question: what is unique to HLBSs and cannot be observed elsewhere? This question is fundamental to understanding which aspects of such systems constitute the hyperlinking component and the form that any such component should take.

All of the models address issues relating to the storage, retrieval and presentation of hyperlinked information. However, it can be observed that an increasing number of HLBSs (e.g., WWW [Berners-Lee, 1991, Berners-Lee et al., 1992], Hyperwave [Dalitz and Heyer, 1996]) adopt a client-server model not subsumed by these models. For instance many are supported by database management systems (DBSs) and rely on the latter’s query language and data management operations [Beeri and Kornatzky, 1990]. The graphical and textual user-interfaces (GUIs) used by HLBSs are often distinct from the hyperlink function [Beatty, 1997, Grauer, 1998]. These observations suggest that functionality is inherited by HLBSs from DBSs and GUIs and as such should not be confounded with P&A issues that are unique to HLBSs.

Finally, no consideration is given in these early models to the fact that users of HLBSs will often have different information seeking goals, background, experiences and levels of understanding of the information embedded within a system.

Following the publication of the Dexter model, several extensions to it have been proposed [Hardman et al., 1994, Garzotto et al., 1994, Grønbæk and Trigg, 1994a, Penzo et al., 1994, Grønbæk and Trigg, 1996]. In addition since the publication of the Dexter model a broad array of models for a variety of aspects of HLBSs have been proposed. These include Microcosm [Fountain et al., 1990, Davis et al., 1992a, Davis et al., 1993], Hy-time [Newcomb et al., 1991, Newcomb and Newcomb, 1992, De Rose and Durand, 1994] and RMM [Isakowitz et al., 1995]. In chapter 3 of this dissertation, a detailed analysis of HLBSs modelling issues is presented.
2.2.3 HLBSs Interaction Issues

The publication of [Conklin, 1987] highlighted the need for further research into many aspects of HLBSs interaction. In particular, the paper emphasised the need to address the need to support user navigation of hyper-networks and the perceived cognitive overload and disorientation suffered by users as a result of the structural complexity of the hyper-network. These issues and the approaches used to address them are now briefly discussed.

It has been argued that the greatest strength and weakness of HLBSs are linked to issues relating to navigation [Jonassen and Grabinger, 1990, Nielsen, 1990, Eklund and Zeiliger, 1996]. The benefits of being able to organise and present information in a HLBS are often compromised by the problems associated with the control of the underlying structure. Hyper-networks have an author (possibly a group thereof). The author has always had complete control of what is presented and how it can be accessed. Although HLBSs inherently provide users with a degree of choice by enabling the user to traverse links freely [Brusilovsky, 1996b], the power to control potential interactions between users of the system and the system itself rests firmly with the author. The author must try to balance their own view of how information within the hyper-network should be organised and accessed with the views and needs of its target users. Traditionally, user needs have only been addressed at the design stage in the development process of constructing HLBSs.

HLBSs are often praised for their ability to make information easily accessible to users via links to information at appropriate points throughout the system. However, it has been observed that users often wish to go directly to a specific unit of information. The link access mechanisms of many HLBSs limit the user to following paths defined by the author through the hyper-network [Karlgren et al., 1994]. This often culminates in the user having to remember which links to follow to get to a particular unit of information. Furthermore, user interaction may be slowed down by the wait for information units to be presented for no reason other than to access a link to some other unit of information on the author’s defined path. This situation may be less significant when the user objectives are not task specific, however, in cases where the user wishes to gather information related to a specific task, the overheads associated with traversing irrelevant links can become high and users can suffer cognitive overload [Thüiring et al., 1995].

It seems clear that not all users of HLBSs can be subsumed into a common model that averages them and that how users wish to access information is task dependent, changes over time and is highly affected by their level of understanding of the subject domain. From this perspective it is reasonable to postulate that users may benefit from personalised interaction with HLBSs.

Many of the current generation of HLBSs have very large hyper-networks. This is due, in part, to the relative ease of extending a hyper-network’s link structure by merely adding
a new link to the existing structure. Openness and expandability raise several issues regarding user interaction with hyper-networks.

As hyper-networks become large, they become increasingly difficult to navigate and to conceptualise in terms of the information available and the context in which a particular unit of information is related to another. A user may often be faced with increasing amounts of information without appropriate navigational or contextual aids to manage this information [Agosti et al., 1995]. Furthermore, HLBSs have been criticised for providing users with too many opportunities to traverse links which are only partially relevant, resulting in cognitive overload [Thüring et al., 1995]. It is also often the case that a user may traverse a link only to be presented with an entirely new area of a hyper-network, which may distract them from their original goal or disorientate them. Therefore, the author must try to reconcile the desire to provide many links to many pieces of information with the risk that users who “want to wander” through the hyper-network find themselves lost [Conklin, 1987].

What can be concluded is that users may benefit from the availability of facilities to not only limit a hyper-network to a subset of relevant information, but to then easily access just the subset when traversing the hyper-network. This process could be described as allowing for the creation of a personalised hyper-network, which may or may not display cohesion to someone other than the person who defined it. These issues are, in part, motivating factors for the research reported in this dissertation.

2.3 Personalisable, Adaptive HLBSs

*Personalisable, adaptive HLBSs* (PA-HLBSs) is an area of research which aims to increase the functionality of HLBSs by making them personalisable [Brusilovsky and Vassileva, 1998, Brusilovsky, 1996b, Milosavljevic et al., 1997, Brusilovsky and Bra, 1998]. Most deployed HLBSs can be described as passive in the sense that they provide the user with the tools and freedom to traverse (or browse) a hyper-network to seek information. During the traversal process all paths through the hyper-network remain static and unchanged. User *model-based adaptive systems* [Kok, 1991, Kobsa, 1993] attempt to guide the user through an information structure (which may or may not be a network) in order to present relevant information, taking into account the user’s history, preferences and information seeking goals. PA-HLBSs research may be viewed as an attempt to combine these two approaches to information retrieval.

Researchers into PA-HLBSs have postulated that such systems may be able to address the following issues:

1. Users may have different levels of experience, background and knowledge and will often have different information seeking goals. By providing users and/or the system
with the ability to personalise the interaction process, it is envisaged that it will become more appropriate to each individual user’s information retrieval needs;

2. By enabling the user and/or the system to restrict the hyper-network to a subset of links and information, it is hoped that this subset will be easier to traverse, thereby reducing the potential for cognitive overload and disorientation;

3. By providing the functionality for users to incorporate their own content, and even links, users will be able to re-design the hyper-network so that it becomes a more tailored, or personalised, version of the original.

Such functionality has only been incorporated into HLBSs to date in very restricted forms and very few systems.

In summary, PA-HLBSs can be broadly described as HLBSs that support user-initiated and system-initiated tailoring. Early work on PA-HLBSs can be found in [Kay and Kummerfeld, 1994, Brusilovsky et al., 1998]. A comprehensive review of methods and techniques used in many recent PA-HLBSs can be found in [Brusilovsky, 1996b].

### 2.3.1 Motivations for Personalisation Requests

The author of a hyperdocument has control over what content is presented and how content may be accessed using links. Traditionally, a user may traverse hyperdocuments using links but may not transform the content or the presentation of links.

HLBSs personalisation is the process by which a HLBS accepts a request from a user to provide a view of an existing hyperdocument that reflects that user’s information goals and history.

A request for personalisation is, therefore, motivated by a user’s information goals and history. A user’s information goals characterise what a user wishes to achieve in terms of information gathering when interacting with the HLBS. A user’s history comprises, in principle, all previous interactions of that user with a hyperdocument. This history, again in principle, characterises the knowledge that a user has of both the hyperdocument and HLBS.

The motivation for a personalisation request, driven by a user’s information goals, can be characterised as comprising of three distinct desires. The desire to control what information is to be provided, how information is provided and where information is to be obtained (see Figure 2.3).
• **What information is to be provided** A user’s wish to control what information is to be provided by the HLBS may encompass controlling what is provided by a hyper-document and/or hyperdocuments of this kind. The scope of this wish covers the content contained within the hyperdocument and/or the presentation characteristics associated with this content. Typically, a user may wish to transform the hyperpage by inserting or deleting content. For example, a user may request that a hyperpage which contains content on all London University colleges be transformed so that the re-specified hyperpage consists of content relating only to Goldsmiths College. A user may also wish to change the presentation characteristics. For example, a user may re-specify that all introductions within hyperpages of this kind be displayed in a specified colour;

• **How information is to be provided** Within HLBSs opportunities to access information are provided to a user by the inclusion of links between hyperpages in a hyperdocument which form paths for navigation. Links may be contextual or non-contextual. Contextual links are links embedded in context within the content of a hyperpage. Such links form “hotwords” in text or “hotspots” in pictures. Non-contextual links are links embedded within a hyperpage independently from the content of the hyperpage. Non-contextual links may be lists, sets of buttons or menus.

A user’s wish to control how information is to be provided by the HLBS may encompass controlling how contextual and non-contextual links are provided by a hyperpage and/or hyperpages of this kind. Typically, a user may wish to transform the hyperpage by inserting, deleting, hiding or annotating contextual and non-contextual links. For example, a user may wish to request that the number of links presented be limited to those deemed relevant or that links are annotated to indicate their level of relevance. Furthermore, a user may wish for links to be sorted. For example, a list of links may be sorted such that the most recently or most frequently traversed links are presented first;
Where information is to be obtained A user’s wish to control where information is obtained may encompass controlling what is obtained by the current HLBS and/or available HLBSs of this kind. The scope of this wish covers data sources employed by the HLBS and the status of the data retrieved. A user may wish to transform a hyperdocument by requesting that only a specified set of data sources be employed by the HLBS when constructing that hyperdocument for display. For example, a user may request that only content from UK university data sources be considered by a HLBS. A user may also wish to specify the status of content retrieved. For example, a user may wish to request that only updated pages are included in hyperpages presented. Such a request would require the HLBS to access many data sources at the time of request to ensure that only the latest content is available when constructing the hyperpage.

The motivation for a personalisation request, driven by a user’s history, can be characterised as comprising three distinct historical knowledge considerations. The historical knowledge a user may have about the content, the hyper-network and the operation of the HLBS itself (see Figure 2.4).

![Diagram](image)

**Figure 2.4: User History Motivations for Personalisation**

**Historical Knowledge of Content** A user’s knowledge about the content of a HLBS may encompass knowledge of the contents of a hyperdocument and/or hyperdocuments of this kind. The scope of this knowledge would cover the content contained within the hyperpage and/or the presentation characteristics associated with this content. Typically, a user may wish to transform the hyperpage by inserting or deleting content which is known to the user. For example, a user may know that a hyperdocument has been structured so that each of its hyperpages contain an introduction, a description of a problem and a practical example of how to solve that problem. On the basis of this knowledge the user may wish to request that only practical examples are displayed when a hyperpage is requested. A user may also have knowledge of the presentation characteristics of the content of a hyperpage. For example, a user may know that a hyperpage contains several animations...
which can be displayed stopped or running. On the basis of this knowledge a user may wish to specify that certain animations on this hyperpage be presented stopped when displayed;

- **Historical Knowledge of the Hyper-network** A user’s knowledge about navigating through a hyper-network may encompass knowledge of navigating through a particular hyperdocument and/or hyperpages of this kind. The scope of this knowledge would cover navigation using contextual links and non-contextual links. Typically, a user may wish to transform a hyperpage by inserting, deleting, hiding or annotating contextual and non-contextual links on the basis of their knowledge of how these links have been included. For example, a user may wish to request that all links that have been previously traversed are hidden until further notice. User knowledge of how non-contextual links have been included may lead to a user wishing to arrange links into certain information units. For example, a list of non-contextual links may be pruned so that those links which have been traversed previously are not presented in the list;

- **Historical Knowledge of HLBSs** A user’s knowledge about the operation of HLBSs may encompass knowledge of the operation of a particular HLBS and/or available HLBSs of this kind. Furthermore, knowledge may extend to knowledge of servers of which HLBSs are a client (i.e., user interface, link and data servers). For example, a user’s historical knowledge of a user interface server may be used as the basis for requests to personalise the operations/functionality provided by the user interface. For example, if a user knew that a user interface server could render a specific mark-up language the user may wish to request that all mark-ups found in a hyperdocument be rewritten in that mark-up language. Existing knowledge of the operations of a data server could be used as the basis for making personalisation requests as to the operation of a data server. For example, if a user knew that a specific data server was very slow in responding to requests from a HLBS then the user may wish to request that the slow data server is not consulted by the HLBS when constructing a hyperpage.

In summary, user and/or system-initiated tailoring within HLBSs may be viewed as the process of accepting a request from a user in order for a HLBS to reflect the user’s information goals and history. To characterise personalisation within HLBSs it is essential to understand the motivations which drive requests for personalisation.

### 2.3.2 Techniques Used to Implement Personalisation Requests

Since research into PA-HLBSs is relatively recent, many of the concepts involved and the techniques used to implement them have yet to be precisely defined. Furthermore, many
of the techniques employed by such systems have been developed as the result of an ad-hoc technology-driven approach. As Brusilovsky states,

adaptation techniques are still unique in the sense that each of them were suggested in conjunction with the development of an adaptive hypermedia system. [Brusilovsky, 1996b]

This approach of concentrating on issues relating to the implementation and use of personalisation techniques in HLBSs has meant that systems and the techniques they employ are difficult to compare in terms of the nature and scope of personalisation they allow for and the implementation strategy used to implement them. Although there have been several experimental investigations to validate PA-HLBSs techniques [Brusilovsky and Pesin, 1995, Clibbon, 1995, Brusilovsky, 1997], in the main this research has been based upon examining the effectiveness and efficiency of a particular technique deployed in a particular HLBS. To date, no large scale empirical studies have been conducted which could provide a prescriptive stance on the deployment of personalisation techniques.

The techniques which have been used to implement personalisation features in HLBSs can be broadly classified as techniques used to tailor content and to tailor links. For example, content personalisation actions could be employed to provide an expert with additional technical information or a new user with additional background information [Beaumont, 1994, Boyle and Encarnacion, 1994]. The purpose of link personalisation actions is to assist users when traversing hyperdocuments.

Five different actions used for content personalisation can be identified in the literature; selective content, goal driven selective content, prerequisite content selection, comparative content selection, and content variation:

- **Selective Content** The aim of selective content is to hide from the user parts of the information about a particular concept which the user has expressed a wish not to see or which the system conjectures as being not relevant to the user’s level of knowledge. Alternatively, additional explanations may be presented to users where appropriate;

- **Goal Driven Selective Content** The aim of goal driven selective content is to hide from a user parts of information about a particular concept which are not relevant to the user’s current goal [Höök et al., 1996]. This approach may be supported by a search facility for a user to express an information goal and have this request act as a filter on the content being presented;

- **Prerequisite Content Selection** Prerequisite content selection is based on prerequisite links between content. The aim is to supplement a request for content with additional
content which describes all prerequisite concepts related to the request which are not sufficiently known to the user. Prerequisite content selection techniques are used in C-Book [Kay and Kummerfeld, 1995];

- **Comparative Content Selection** Comparative content selection is based on similarity links between content. The aim is to supplement a request for content with similar content, together with descriptions of similarities and differences between the original content requested and the additional content provided;

- **Content Variation** The aim of content variation is not merely to hide or show some portion of content, but to provide the user with a variant of the content which the system conjectures to be appropriate. Such an approach requires the HLBS to store several variants of content about a concept. This action is used in Anatom-tutor [Beaumont, 1994] and Hypadapter [Hohl et al., 1996].

Five different actions used for link personalisation can be identified in the literature; directed guidance, hiding and blocking, annotation, sorting and map adaptation:

- **Directed Guidance** The aim of directed guidance is to provide a user with navigational support by suggesting a link to follow. This approach requires the system to conjecture and then make a recommendation as to the next best link to follow based on knowledge about the user. At its extreme, directed guidance can take the form of a dynamically generated next link. This link appears in addition to those within the content being viewed and can be used as an alternative mode of navigation. Directed guidance can be used with both contextual and non-contextual links. Approaches to directed guidance are found in [Brusilovsky and Pesin, 1994];

- **Link Hiding and Blocking** Link hiding actions have been used to limit the number of navigation opportunities presented to the user to those deemed relevant. Link hiding attempts to decrease cognitive overload by letting the users concentrate on analysing the most relevant links. Links which are not relevant to the user are hidden. The decision as to which links are relevant is made according to the user’s information goals and history. Various approaches to link hiding have been implemented in Hynecosum [Vassileva, 1996] and the PUSH system [Höök et al., 1996]. Further research into link hiding can be found in [Brusilovsky and Pesin, 1995]. Link blocking is a variant of link hiding by which the number of navigation opportunities a user may take are limited by making certain links inactive. Those links that are inactive may not be traversed by the user although the user is aware of their existence;

- **Link Annotation** The aim of link annotation is to inform the user about the type of information which would be presented if a link is followed. Link annotation can be used to represent link relevancy or to reflect levels of user knowledge about the information which would be displayed. Annotations may take the form of additional text or visual clues [de La Passardiere and Dufresne, 1992, Brusilovsky and Pesin, 1995].
• **Link Sorting** Link sorting actions present links according to their relevance to the user’s information goal and history. The applicability of link sorting is limited in that it can only be used where links are non-contextual. Research has indicated that link sorting can reduce navigation time when hyperpages contain many non-contextual links [Kaplan et al., 1993]. Link sorting may result in the order of links within a hyperpage changing each time the user requests it. This situation, it is argued, may lead to increased levels of cognitive overload. Link sorting is used in HyperMan [Myka et al., 1992, Mathe and Chen, 1996] and HYPERFLEX [Kaplan et al., 1993];

• **Map Adaptation** Since HLBSs interaction is primarily based on navigating a hyper-network by traversing links, global and local maps might help. Maps depict paths through the hyper-network aiming to provide users with a visual representation of the domain of concepts within the hyper-network. Furthermore, maps may themselves have embedded links which may be traversed by users to navigate from some place within the hyper-network to another.

   Map adaptation is concerned with providing users with various features to tailor the form of global and local HLBSs maps. Map adaptation may involve highlighting paths through a map, hiding parts of the map and even specifying levels of detail shown on maps [Botafogo et al., 1994, Zyryanov, 1994, Nielsen, 1995].

It can be argued that although the work mentioned above exemplifies how personalisation may be implemented in specific systems, it fails to characterise the scope of possibilities for personalisation within HLBSs. Understanding this scope should clarify the functionality which a user may wish to utilise and which personalised HLBSs could provide. Such a clarification is a crucial motivation for the work reported in this dissertation.

### 2.3.3 User Modelling in Personalisable, Adaptive HLBSs

System-initiated tailoring within the context of HLBSs refers to the interactive process of transforming a HLBS through the use of an *adaptive mechanism*. This mechanism is used to:

1. develop a model of each user;

2. decide on the basis of that model and a prescriptive theory of what may be useful for a user in different circumstances, what personalisation actions the user might have taken;

3. to initiate actions on behalf of the user.

More generally, the process may be described as: collecting data about the user, processing the data to build or update the user model and applying the user model to provide system-initiated tailoring.
A user model stores an individual user’s information goals and history. Such a model is the basis upon which users can provide input into the personalisation process. User models may take many forms [Brajinik et al., 1987, Kay and Kummerfeld, 1994, Beaumont, 1994, Brusilovsky and Vassileva, 1998, Karlgren et al., 1994] and many different techniques have been used to acquire information about users [Kok, 1991, Kobsa, 1993].

A specific concern when incorporating user modelling techniques into PA-HLBSs is the sharing between the user and the system of the process of constructing and then updating the user model. Broadly, in PA-HLBSs, the system is always responsible for applying the user model to provide system-initiated tailoring. However, the stages of collecting data about the user and processing the data to build or update the user model may be shared. A comprehensive review of issues related to the sharing of responsibilities can be found in [Malinowski et al., 1992].

User models may be grouped into two broad categories, empirical quantitative models and analytical models [Brajinik et al., 1987]. Empirical quantitative models are generally based on an abstract formalisation of general classes of users. The foremost approach is to consider knowledge about the user explicitly during the design stage and then hard-wire these considerations into the system. This approach is often seen in conventional help systems.

Analytical cognitive models attempt to simulate the cognitive user processes that take place during interaction with the system. These models incorporate an explicit representation of user knowledge and use this as a basis for analysis of specific traits of users.

The majority of PA-HLBSs employ one of the following strategies to incorporate user modelling into their adaptive mechanisms: stereotyping, preference analysis or activity analysis:

- **Stereotyping** Stereotyping requires the surveying of potential users in order to elicit a set of typical user information goals and history. This information is then used to form one or more “stereotypical” user models of particular classes of user. Stereotyping is often used in conjunction with an overlay model whose function is to “overlay” a stereotypical user model upon the hyper-network. Typically, for each unit of information in the hyper-network an individual overlay model stores some value which is an estimation of the user’s knowledge about this unit. More generally, overlay models are represented as a set of “concept-value” pairs [Brusilovsky, 1996b]. Overlay models were originally applied to the area of intelligent tutoring systems [Frasson et al., 1992, Greer and McCalla, 1993].

Stereotyping relies upon the designers of PA-HLBSs to determine the stereotypical groupings, as such, it has been argued that stereotypical modelling cannot be anything more than an educated guess of user needs [Vassileva, 1996];
• **Preference Analysis** Preference analysis can be described as a “glass box” approach to the construction of a user model [Höök et al., 1996]. Each user’s model is entirely created by the user and no interpretation of the user’s model is undertaken by the designers of the PA-HLBSs. On the surface this approach may appear to be too simplistic and general, however, upon further examination of the difficulties associated with approximating user information goals, histories and potential requests [Karlgren et al., 1994], in many cases a user model specified by an individual user will more accurately reflect a user’s wishes than a stereotypical model. A further point to note is that many techniques used in PA-HLBSs require a more finely tuned user model than those which can be achieved through stereotyping alone;

• **Activity Analysis** Activity analysis is a systems-based approach in which user interactions with the PA-HLBSs are recorded and analysed. Activity analysis usually involves the system monitoring the behaviour of the user’s interaction with the system in terms of how they traverse hyperdocuments. Based upon this information the system attempts to infer a user model which the user’s interactions appear to express. It must be noted that this approach is both system-centred and retrospective. The user is, at best, only informed that a transformation has taken place. It is rare to find systems in which the user is told what decision has been made and how that will affect future interactions.

Although stereotyping, preference analysis and activity analysis have been employed individually to realise system-initiated tailoring, it is often the case that a combination of approaches are used. Such combinations are referred to as collaborative user models [Kay, 1995]. For example, a PA-HLBSs may use a stereotypical user model as a start point, but then allow users to re-specify it according to their individual needs. Furthermore, a system may track user interaction as in activity analysis but then inform the user of its conjecture and allow the user to override its decision where appropriate.

In summary, a user model may provide the means for a user to feed preferences into the personalisation process when it is system-initiated. The model reflects a user’s goals and history. User modelling generally involves elicting details from the user overtly or covertly. These details are then used by the adaptive mechanism to perform system-initiated tailoring.

In this dissertation, adaptation is synonymous with system-initiated tailoring and is, in principle, as expressive as user-initiated tailoring and requires no technologies other than those involved in user modelling and in decision making from a user model.
2.3.4 Applications of Personalisable, Adaptive HLBSs

The predominant application domain for PA-HLBSs to date has been education [Kobsa et al., 1994, Brusilovsky et al., 1996a, Skillicorn, 1996]. Such systems generally encompass relatively small bodies of information resulting in relatively small hyper-networks. Users usually have a specific goal, viz., to learn a subset of the information contained within the hyper-network. The application of personalisable and adaptive methods and techniques is used to support each user's knowledge acquisition tasks on an individualised basis. An important requirement of such systems is an ability to accommodate different levels of user knowledge of the subject matter.

In the educational domain, PA-HLBSs may be used to suggest to the user appropriate routes through the hyper-network and to indicate, or even restrict, users to specific subsets of information on an individualised basis [Brusilovsky, 1996a].

PA-HLBSs have also been applied to the domain of on-line information systems [Boyle and Encarnacion, 1994, Hohl et al., 1996, Höök et al., 1996]. Such systems are generally used as reference resources, such as help systems, encyclopedias and user manuals. On-line information systems tend to encompass several subject domains and their size and complexity can be relatively large. Due to the tendency of on-line retrieval systems to contain a wide variety of information, it is envisaged that being able to restrict this information to a subset of relative information may be useful in achieving efficient information access. Furthermore, users of on-line information systems may well have a specific concept in which they are interested and little time to examine the hyper-network searching for it. It is thought that the ability to quickly traverse the hyper-network by first personalising it may enhance such systems' usability.

A new and interesting application area for PA-HLBSs is that of organisational information systems. Such systems tend to be large in size and complex and aim to incorporate all relevant information about an organisation. Examples of organisations which use such systems include hospitals, government institutions and large businesses. Organisational information systems were usually originally developed as a loosely related set of databases, each of which stored information relating to a particular aspect of the organisation. Recently, the linking facilities of HLBSs such as Lotus Notes [Moore, 1995, Dale and Opyt, 1996] and the WWW have begun to be used as the “glue” used to link these different information sources together.

Large organisations are usually comprised of different groupings of personnel who may themselves be members of several different groups. Furthermore, the composition of these groups may change frequently. An organisation’s personnel may need to communicate with and share information with many of their colleagues in many different groups. Each group in an organisation is usually defined by the function it performs or by a specific activity that the group is engaged in. Thus the groups and the individual members of groups...
will have different information requirements. It has been postulated that PA-HLBSs may provide the flexibility to allow an organisation’s personnel groupings, or the organisation itself, to restrict the information that the group or individual may access to a subset which is appropriate for their specific function or activity.

A new class of information system which has become an application area for PA-HLBSs research is public HLBSs, (i.e., large-scale library systems, on-line shopping facilities and public information systems). Public HLBSs are characterised by the fact that the user base, in both nature and numbers, is unknown. Furthermore, the hyper-networks which underpin public HLBSs tend to be large in size. A crucial factor in the design of public HLBSs is the approximation the designer must make as to who will access the public HLBS. If the designer’s approximation is inaccurate then many of its potential users may assume that the system does not cater for their needs and may therefore decide to use an alternative source of information, or in the case of on-line shopping, another supplier. PA-HLBSs techniques may be used to endow such public HLBSs with facilities to allow unknown users to customise their access to public information systems and to assist in their navigation through such systems. Furthermore, the availability of personalisation facilities within such systems may be deployed in such a manner as to gather information about its users which may subsequently be used in the approximation process when refining the existing public HLBS.

Other application areas include the creation and management of personalised views of information spaces [Egov et al., 1994, Dieberger and Bolter, 1995], on-line help systems [Gonschorek and Herzog, 1995] and general office documentation systems.

2.4 Summary Discussion

The predicted growth in WWW use is likely to make the deployment of HLBSs even more attractive. In particular, there has been an upsurge of interest in HLBSs as a result of the perceived effectiveness of WWW browsers [Duval and Main, 1994, Beatty, 1997, Grauer, 1998] in enabling multimedia information on the Internet to be accessed via hyperlink-based interaction in a much wider scale than envisaged a few years ago. However, the development activity that resulted from this increased level of interest has been driven primarily by technological innovation and constrained only by technical feasibility. As a consequence, implementations have displayed limited compatibility and a shared set of design principles has yet to emerge. At a more fundamental level, and in a predictable manner, this hands-on approach has not been accompanied by a more abstract, more formal understanding of many issues raised by HLBS’s technological advances and shortcomings.

A significant factor limiting the effectiveness of many of the current generation of HLBSs is their lack of support for user-specific interaction in the form of personalisation.
Many of the current generation of HLBSs can be characterised as static, in that they provide all users with an identical, predefined hyper-network. This means that crucial user-specific choices are limited to which link to traverse and when to do so. Since users often have different goals, backgrounds and experience and since the very dynamics of the learning process causes users' requirements to change over time, it is widely believed that unless HLBSs cater for personalisation and adaptivity actions, their effectiveness is likely to hit a lower ceiling than previously anticipated.

Several HLBSs have been proposed which aim to be dynamic at the level of the information they deliver (see, e.g., [Brusilovsky, 1996b]). However, in the majority of such systems, support for user-specific interaction has not been motivated in the light of reasoned principles and hard evidence. At the appropriate abstraction level, there is still insufficient understanding of where and how, within HLBS, personalisation and adaptivity should be designed and when and how, at the time of interaction, it should be manifested. It can be argued that the models for HLBSs proposed to date are not an adequate foundation for such an investigation. There have been many proposals as to which personalisation and adaptivity actions a HLBS could, or should, support [Brusilovsky, 1996b], but it appears that little work has been done on situating these extensions against the background of an abstract model of PA-HLBSs.

At present, therefore, the hypothesis that the effectiveness of HLBSs would be increased if only they possessed the ability to adapt to the needs of individual users has not been conclusively tested because the means for this are not yet available.

In the research reported in this dissertation it is argued that to test the hypothesis that personalisation and adaptivity are effective, several preliminary goals must be achieved. Firstly, it is necessary to characterise, in a principled, systematic manner and as comprehensively as possible, the space of possibilities for personalisation and adaptivity actions. Secondly, one or more prototype HLBSs must be built which implement (possibly a subset of) the personalisation and adaptivity actions thus identified. Thirdly, metrics must be chosen to measure the effect of personalisation and adaptivity actions. Fourthly, prototype HLBSs must be deployed in empirical investigations with a view towards accurately and exhaustively measuring, for each chosen metric, the effect of personalisation and adaptivity actions. This dissertation addresses the first of the above needs, and in part, also the second. The third and fourth are beyond its scope.

This chapter aimed to place the research reported in context by summarising the evolution of the major trends and issues related to PA-HLBSs. Chapter 3 of this dissertation presents a detailed discussion of PA-HLBSs modelling issues. This chapter also introduces a formal model for personalisable, adaptive hyperlink-based interaction which is the major contribution of the research reported in this dissertation.
Chapter 3
A Survey of Hyperlink-Based Modelling Issues

This chapter provides a detailed analysis of PA-HLBSs models and modelling issues. Clearly, not all models proposed can be discussed here. The criteria for inclusion is either that the model clearly reflects an important modelling trend or that the model possesses specific characteristics which may be viewed as important in the context of this dissertation. This analysis aims to promote an understanding of the scope of hyperlink-based models and their form. Furthermore, it is hoped that this will, in turn, place in context the formal model for PA-HLBSs, which is the major contribution of the research reported in this dissertation.

The chapter begins with a brief introduction to hyperlink-based modelling issues. Following this, several models of HLBSs are discussed. Models are categorised into the following: those that model possible instantiations of HLBSs; those that model the integration of hyperlink-based technologies with other computing environments; and those which aim to model user interaction issues, such as personalisation and adaptivity. Finally, an introduction to the model for personalisable hyperlink-based interaction, which forms the main contribution of this dissertation is given.

3.1 An Introduction to Hyperlink-Based Modelling

Hyperlink-based modelling is concerned with creating models for the purpose of managing and representing hyperlinked data, information and knowledge. The aim of such research is to develop models which may be used to allow users to manage hyperlinked data, information and knowledge in an effective and efficient manner.

In the following section, a brief introduction is given to the modelling of hyperpages, links, hyper-networks and HLBSs requirements.
3.1.1 Modelling Hyperpages

Hyperpages, which form the “nodes” in a hyper-network, may consist of both digitally represented static media, such as text and graphics, and dynamic, time-based media, such as video, sound and animation. Broadly, the operations which can be performed on hyperpages consist of presentation (e.g., the rendering of hyperpages or the displaying of video), manipulation (e.g., the editing or personalisation of a hyperpage or one of its component parts) and analysis (e.g., the conditional selection or ordering of a subset of hyperpages or their component parts).

In the majority of HLBSs, hyperpages are viewed as static units whose content is “hard-wired” at the point of design. As a result, all content is explicitly represented by the author and the system is merely responsible for retrieving that content so that some user interface may present it. As well as content, a hyperpage may also contain instructions on how to present content (e.g., mark-up tags in HTML).

Hyperpages may also be viewed as dynamic in the sense that a hyperpage can contain specifications of what content it should have and from where this content is to be sourced. Furthermore, such specifications may cater for the inclusion of queries to retrieve links from link databases. With dynamic hyperpages, the author is responsible for the design of specifications and at run-time (i.e., when a hyperpage is requested or a user begins a session using the HLBS) specifications are composed and then resolved by the system into the content and links which they denote. The hyperpage, complete with content and links, is then presented. Such a dynamic approach has several potential advantages including:

1. It is possible to include content that is held remotely from the HLBS itself and which the author may have little or no direct control of;

2. It is possible to interfere with the contents of a hyperpage by merely rewriting its specification, rather than rewriting the content part of the hyperpage;

3. It is possible to easily incorporate the latest version of content as it is naturally fetched at run-time;

4. It is possible to utilise the benefits and functionality of recognised data management technologies, such as database systems, for the management of content;

5. It is possible to include in hyperpage specifications the details of how content should be included in a hyperpage and under which circumstances;

6. A dynamic approach can facilitate information re-use and aid information management, as it may be possible to store only one copy of a unit of content which may then be referenced by more than one hyperpage.
The major disadvantage of a dynamic approach is that the time taken to resolve a specification into content may result in indeterminate delays when a user requests a hyperpage. However, if all hyperpages are resolved at the start of a session and the content held locally then such delays may be minimised. Another disadvantage is that, content cannot be guaranteed by the author as it is realised at run-time. In this dissertation, hyperpages are modelled as dynamic specifications and personalisation is modelled as the rewriting of such specifications. Figure 3.1 depicts static and dynamic models of hyperpage retrieval.

![Image](image.png)

Figure 3.1: Static and Dynamic Models of Hyperpage Retrieval

### 3.1.2 Modelling Hyperlinks

Hyperpages are inter-related (logically and structurally) using sets of pointers, commonly referred to as links or hyperlinks. A link connects a source anchor to a destination anchor. Generally, an anchor comprises a name and an offset (or position) within a hyperpage. However anchors may also be realised as the result of interpreting a specification that describes the attributes of the anchor rather than name and position only. Anchoring is the mechanism for referring to a part of a hyperdocument [Halasz and Schwartz, 1990,
Halasz and Schwartz, 1994]. Anchors may be visualised as a piece of text, a diagram, video or a hyperpage which when traversed (e.g., by a mouse click) displays to the user the hyperpage whose name is associated with the destination anchor.

Broadly, given two hyperpages a link can be classified as providing the following types of traversal possibilities:

- *document offset to document offset*: a traversal possibility from some place in document one to some place in document two;
- *document offset to document*: a traversal possibility from some place in document one to document two;
- *document to document offset*: a traversal possibility from document one to some place in document two;
- *document to document*: a traversal possibility from document one to document two.

Many kinds of links have been proposed and implemented. These include simple links [Conklin, 1987], generic links [Fountain et al., 1990, Davis et al., 1992b], scripted links [Rizk and Sauter, 1992], functional links [Ashman and Verbyla, 1994] and adaptive links [Miller and Wantz, 1998b]. Figure 3.2 depicts some of these possibilities.

The most common form of link, a simple link, [Conklin, 1987] provides a one-to-one mapping between hyperpages. Such links may be embedded [Brown and Brown, 1995] or held separately from the hyperpage in a link database (referred to as a linkbase) [Davis et al., 1994].

In [Fountain et al., 1990] the notion of a generic link is introduced. Generic links allow for the connection of any occurrence of a source token (source link name) in a hyperdocument to a single destination anchor. As such, generic anchors provide a many-to-one mapping between source tokens and a destination hyperpage. The many-to-one mapping is realised via an intermediate linkbase that holds both the source and the destination anchor details. Such links are said to be “freely defined” [Brown and Brown, 1995]. When a source token is traversed, the HLBS consults the linkbase to compute the destination anchor and complete the traversal into the target hyperpage. Benefits of generic links include a much lower level of authoring effort and a potential reduction in the number of actual links that need to be stored.

An early approach to incorporating a dynamic element into links was the notion of the scripted link [Rizk and Sauter, 1992]. Broadly, scripts of information are attached to links so that the destination of a link is computed by interpreting a script. Such scripts are used to extend the behaviour which a link may exhibit. The applicability of scripts is often limited by the fact that such scripts are often system-specific. However, one of the
earliest commercial HLBSs, Guide [Hershey, 1987] provided facilities for the incorporation of scripts which were not system-specific and could be used to calculate destination anchors and also launch third party applications.

A functional link is a link in which both the source anchor and destination anchor are computed by arbitrary functions at navigation time [Ashman and Verbyla, 1993, Ashman and Verbyla, 1994].

Such a function may utilise environmental factors such as navigational history, navigational scope and security requirements. A functional link consists of a point of connection (a point within a document within which a link can be activated) and an environment (a set of factors which influence the interpretation or availability of the link). A link is defined by a function pair, consisting of a source predicate and a resolution function. The process involves determining whether a candidate connection point is actually a source. The source determination function is therefore a predicate. The resolution of the destination set of a link maps the known source onto the subset of possible connection points which constitute the link’s destination.

Using the functional model, it is possible to represent nearly all types of link in a single framework. At present, few HLBSs incorporate the notion of functional links, however, one area in which such links have been investigated is that of PA-HLBSs.

Functional links have the ability to allow the author to tailor the navigational possibilities available to the user at any given point. The COOL link model [Miller and Wantz, 1998a, Miller and Wantz, 1998b] is an example of the use of functional links for this purpose. A COOL link is a multi-ended computed link for the WWW. A COOL link contains the following: the implementation of the link itself; meta-data about the linked resources; external input features such as user profiles and link computation algorithms. COOL links refer to an unordered collection of resources when they are activated. When a COOL link is activated each resource in the collection is evaluated and compared against a separate set of input parameters (e.g., user profile) and the “best fit” resource is returned. Each distinct resource in a COOL link collection is assumed to provide different benefits to different users.

In summary, the role of a link is to represent a relationship between two units of information. Relationships need not be fixed and can be determined as a result of a computation. Strategies for the design and implementation of links vary widely. Therefore, it may be argued that any model of HLBSs should constrain as little as possible the types of relationships (i.e., link types) which a hyperdocument author can create [DeRose, 1989].

In the model proposed in this dissertation the hyperpage specifications modelled cast the notion of a link at a level of abstraction such that all link types described may be incorporated in a non-destructive manner. Links could be made dynamic by a simple generalisation of the framework used to provide dynamic content, but, in this dissertation,
links are kept static.

### 3.1.3 Modelling Hyper-networks

The underlying structure of a hyper-network conforms to a structural model. This model helps the designer of the hyperdocument to clarify the role of information units and the inter-relationships between them. A hyper-network, for example, partially defines the navigational possibilities available to the user. The choice of structure for a hyper-network will ultimately be based on the level of accessibility and manipulation of information the designer wishes to provide.

In the context of HLBSs, the basic approach to designing structures may be described as follows:

1. *Deconstruction*, through which information is partitioned into atomic units (i.e., text, graphics, video). A unit is atomic if further partitioning will render the information meaningless, at least in the designer’s intended context;

2. *Reconstruction*, through which atomic units are grouped into appropriate complex structures, e.g., hyperpages;

3. *Association* through which hyperpages and their component parts are interlinked to create complex structures which support navigation and information access, namely hyperdocuments.

Three common structures are sequence, tree and graph [Lowe and Hall, 1999]. A sequence structure is one in which units of information are linked together in a linear sequence. A
linear approach may be used to constrain a user’s navigational possibilities (e.g., to create guided tours). Furthermore, sequence structures may be used to reflect the linear structure of the original, paper, document.

Tree structures may be used to replicate the hierarchical structure found in paper-based documents such as books and technical reports. Typically, such structures offer a table of contents or an index.

A graph structure is generally used to reflect semantic links between information units. Such links are associative and are generally deployed in a non-sequential manner.

It may be the case that a combination of information structures underlie particular parts of a hyper-network resulting in a hybrid structure. Figure 3.3 depicts various information structures used in the construction of hyper-networks.

![Hyper-network Information Structures](image)

**Figure 3.3: Hyper-network Information Structures**

### 3.1.4 Requirements for HLBSs Models

Layered on top of a hyper-network’s structural model is a model of hyperlinked functionality. Such models may be used to develop HLBSs applications. When analysing models of hyperlink-based functionality it may be useful to understand which issues give rise to requirements for such models.

In 1988, Halasz presented an analysis of NoteCards, one of the most popular of the first generation of commercial HLBSs [Halasz, 1988]. It argued that several problems in NoteCards reflected fundamental weaknesses in the hyperlink model around which it and other HLBSs were built, and that precisely these fundamental weaknesses should form the research agenda for the subsequent generation of HLBSs. Many of the issues highlighted remain unresolved and several may be regarded as motivations for the research reported
in this dissertation. A brief summary is now given of the limitations and possible solutions presented by Halasz:

1. **Search and query in hyper-networks** The NoteCards experience suggested that the provision of traversal possibilities via links alone was not sufficient for all information seeking tasks. It is argued that effective access to information stored in a hyper-network requires query-based access to complement navigation.

In the model proposed in this dissertation a formal language is defined for the personalisation of hyper-networks. The language provides the functionality of query-based access by allowing users to rewrite hyperpage specifications that used to form hyper-networks. This approach can be contrasted with that proposed by Halasz in which only DBMSs are viewed as providing such functionality.

2. **Composites** The basic node/link model upon which many HLBSs are built is severely restricted by the lack of sophisticated composition mechanisms (i.e., ways of representing and dealing with groups of hyperpages and links as unique entities separate from their components). It is argued that to design composition mechanisms appropriate for inclusion within HLBSs, the following questions and issues need to be addressed:

- Can a given node (hyperpage) be included in more than one composite?
- Do links necessarily refer to a node per se, or can they refer to a node as it exists within the context of a given composite? If the latter is possible, what does it mean to traverse a link?
- How does one handle versions of composite nodes? Does a new version of the node necessarily imply a new version of the composite?
- Should composites be implemented using specialised links, or is a whole new mechanism necessary?

In the model proposed in this dissertation, an approach is presented for the inclusion of composite objects (information units). This approach centres around devising and formally defining hyperpage specifications as sequences of discrete parts, referred to as chunks. Each chunk may contain content and specifications of how to retrieve content, presentation instructions and links to other chunks. The approach taken allows the model to address all of the questions described above;

3. **Virtual structures for dealing with changing information** It was observed that HLBSs tend to have difficulty incorporating rapidly changing content. This arises from the essentially static and fragmentary nature of their hyper-networks. In particular, networks generally cannot reconfigure themselves in response to changes in the content they contain. This lack of dynamic mechanisms limits their utility in many information seeking tasks. Halasz proposes a relaxation of the overly static nature
of hyper-networks. Specifically, it is argued that hyperlink models need to be augmented with a notion of virtual or dynamically-determined structures.

The approach taken to answering this challenge in this dissertation is, rather than to specify content within hyperpages, to define the content of each hyperpage in the hyper-network as a specification which, when resolved, allows content to be dynamically assigned;

4. **Computation in/over hyper-networks** Halasz argues that the integration of HLBSs and other technologies is a natural progression path which may increase the potential of HLBSs as effective information retrieval systems. It is speculated that HLBSs could support computational engines as separate external entities that create, access and modify information via a programmatic interface. Alternatively, HLBSs models could be designed to incorporate an active computational component that automatically processes (makes inferences from) the information stored in its hyper-network. One such approach is described in [Ashman and Nürnberg, 1999] where the notion of a HLBS having a degree of “structural awareness” is described;

5. **Extendibility and tailorability** To HLBSs, all hyperpages and their links are essentially the same: objects to be stored, retrieved and displayed. To the user, hyperpages are filled with meaningful contents and organised into meaningful structures. HLBSs cannot operate directly on this meaning. They simply provide a collection of tools to manipulate the hyper-network in meaningful ways. The challenge is to design mechanisms to allow the tailorability of the hyper-network by users. Such mechanisms should allow tailorability to be achieved with a small amount of work and not interfere with facilities that allow expert programmers to make major changes to the HLBS.

In the model reported in this dissertation, an approach to empowering users with facilities to tailor hyperpages and their links is proposed. This approach centres around providing users with the ability to tailor hyperpage specifications on an individualised basis. Such tailorability allows users to personalise what content is included in hyperpages and how that content is presented;

6. **Support for collaborative work** It was suggested that HLBSs should possess support for collaborative work. Such support should not only allow for simultaneous multi-user access to a common hyper-network, but also provide support for the social interactions involved in collaboratively using a shared hyper-network. In HLBSs, users and designers need to understand not only the content of the hyper-network, but also the accepted procedures to be used to modify that hyper-network. This could be achieved by extending the standard technologies for shared databases (e.g., transactions, concurrency control and change notification) to address HLBSs specialised requirements;

7. **Versioning** Mechanisms for versioning allow users to maintain and manipulate a history of changes to their hyper-network. Furthermore, versioning allows users to
simultaneously explore several alternative configurations for a single hyper-network. It is argued that a HLBS that lacks versioning mechanisms is severely limited in the range of applications it can support;

In summary, Halasz proposed that HLBSs models should reflect the dynamic and wide ranging information retrieval needs of their users. Furthermore, it is speculated that allowing users of HLBS to tailor the information presented to them may increase the usability of such systems as instances of information retrieval systems.

Although the work reported in this dissertation shares similar objectives to that of Halasz it differs in several important respects:

- Halasz views the functionality required to search and query hyper-networks as being supplied by building into HLBSs querying facilities found in many DBMSs. Although the work reported here sees DBMSs as an important technology, of which HLBSs tend to be a client, it views searching and querying capabilities as arising from the ability to allow users to select and rewrite hyperpage specifications.

- Halasz argues that HLBSs should provide facilities to support collaborative work by including features to allow the control of transactions between authors and to provide concurrency control of data. In contrast the work reported in this dissertation views such functionality as being outside the scope of the HLBS proper and being the responsibility of the data servers that may be queried by the HLBS proper.

- The approach proposed by Halasz promotes the idea of HLBSs being extended to incorporate DBMS and UIS functionality. In contrast, the approach reported in this dissertation views HLBSs as operating in a client-server environment in which such functionality is inherited as and when required from server technologies and is not incorporated into the HLBS itself.

### 3.2 Modelling Instantiations of HLBSs

At present, no model has been accepted as standard by the entire HLBSs community. Furthermore, no model has been proposed that can be comprehensively used to describe all existing HLBSs.

Early research into HLBSs models aimed to address the question of which characteristics were common to HLBSs and how structural, behavioural and navigational characteristics could be best represented. Such efforts could broadly be described as modelling instantiations of HLBSs.

Two of the most influential models proposed are the Dexter model [Halasz and Schwartz, 1990]
and the Hypertext Abstract Machine (HAM) [Campbell and Goodman, 1987]. These models distinguish three levels within a HLBS:

1. **Presentation level** Graphical or textual user interface;
2. **Hypertext Management Level** A database of nodes and links;
3. **Database Level** Storage of shared data and provision of network access.

The drawing of these distinctions was important because it showed it was possible to model particular levels within HLBSs separately from each other. Furthermore, it became possible to speculate that the functionality provided at each level could be delegated with the HLBS providing the communication controls and protocols required to interact between levels.

A brief analysis is now provided of these models and their various contributions outlined.

### 3.2.1 The Hypertext Abstract Machine (HAM)

The HAM [Campbell and Goodman, 1987] was a milestone in HLBSs research because it was the first attempt to define an abstract model of HLBSs. The HAM was described as follows:

The HAM is a transaction-based server for a hypertext storage system. The server is designed to handle multiple users in a networked environment. The storage system consists of a collection of contexts, nodes, links and attributes that make up a hypertext graph. [Campbell and Goodman, 1987]

The HAM is based on an architecture in which a HLBS is partitioned into three levels: the presentation level, responsible for presenting information to users; the hypertext abstract machine, where the fundamental structure of the hyper-network is modelled; and a database level, which manages data storage, access and security.

The definition consists of a description of HAM objects and the operations that can be applied to them. The HAM does not describe a HLBS completely, rather it describes a lower level machine tied closely to the storage system and loosely to applications and user interfaces (see Figure 3.4).

The HAM storage model is based on five types of objects; graphs, contexts, nodes, links and attributes. The HAM maintains a history for these objects and allows selective access through a filtering mechanism. Each of these objects is now briefly described:
- **Graphs** contain all the information regarding a general topic. A graph contains one or more contexts;

- **Contexts** partition data within a graph. Each context has one parent context and zero or more child contexts. Contexts can be used to provide private workspaces;

- **Nodes** contain arbitrary data (text or fixed-length binary blocks). A node may be classified as either archived, non-archived or append only. This classification is used to define how a node may be updated and which nodes are available to users;

- **Links** are used to define a relationship between a source node and a destination node and can be followed in either direction;

- **Attributes** can be attached to contexts, nodes or links. Attributes/value pairs give the semantics to HAM objects. They can represent application specific properties of objects or contain information that further describes an object.

![Figure 3.4: The Layers of the HAM](image)

The operations which can be applied to HAM objects include those to create, delete, destroy, change, and retrieve such objects. The HAM also specifies a filtering mechanism to allow subsets of HAM objects to be extracted from large graphs.

In summary, the HAM model may be viewed as not only the first abstract model of HLBSs, but also still one of the most relevant. The HAM viewed HLBSs as being tightly coupled to host file systems and loosely coupled to a set of applications and user interfaces. This view makes the HAM appropriate for modelling OHLBSs architectures of WWW-based applications where such a binding relationship is crucial. Furthermore, it can be seen...
from the specification of the HAM that its authors realised the need for features to allow personalisation of hyper-networks. The filtering mechanism described in the HAM allows subsets of HAM objects to be extracted from large graphs, thereby allowing a user to specify a subset of objects to be presented.

3.2.2 The Dexter Hypertext Reference Model

The Dexter Hypertext Reference Model proposed in 1989 [Halasz and Schwartz, 1990, Halasz and Schwartz, 1994] was the result of two small workshops on hypertext held in October 1988\textsuperscript{1}. The model is an attempt to capture, both formally and informally, the important abstractions found in a wide range of existing and future HLBSs\textsuperscript{2}. The goal of the model is to provide a principled basis for comparing systems as well as for developing interchange and interoperability standards. The motivations for the model are perhaps as important as the model itself. In devisualising the model, its authors sought to provide a principled basis for answering the following questions:

1. What do HLBSs have in common?

2. How do HLBSs differ?

3. In what way do HLBSs differ from related classes of systems?

Although HLBSs had, at an abstract level, several common features (i.e., a network of information containing nodes interconnected by relational links), implemented systems differed greatly. Another motivation for the Dexter model was the need to find a common terminology for the field.

The model divides a HLBS into three layers; the storage layer, which describes a network of nodes and links; the within-component layer, which addresses the content and structures within hyperdocuments; and the run-time layer, which outlines mechanisms to support user interactions (see Figure 3.5). The model is formalised using Z [Woodcock and Davies, 1996].

The main focus of the model is on the storage layer, which describes a database composed of a hierarchy of data-containing components which are interconnected by relational links. The components themselves are treated as generic containers of data.

The within-component layer is concerned with the contents and structure of components of the hyper-network. Due to the range of possible content/structure that can be included

\textsuperscript{1}The following people attended these workshops and were instrumental in the development of the model; Rob Akscyn, Doug Engelbart, Steve Feiner, Frank Halasz, John Leggett, Don McCracken, Norm Meyrowitz, Tim Oren, Amy Pearl, Catherine Plaisant, Mayer Schwartz, Randy Trigg, Jan Walker and Bill Wieland.

\textsuperscript{2}Among the systems discussed were Augment, Intermedia, Hypercard, Hyperties, KMS/ZOG, Neptune/HAM, NoteCards and the Sun Link Service.
in a component, this layer is purposely not elaborated upon within the Dexter model. Instead, the Dexter model treats the within-component structure as being outside the scope of the model. However, the model describes an interface mechanism for addressing locations or items within an individual component. This mechanism known as anchoring, provides a clean separation between the storage and within-component layers. Furthermore, anchoring provides functionality that allows links to be made not only between documents, but also between parts of documents.

The storage and within-component layers treat a hyper-network as an essentially passive data structure. However, HLBSs in general offer tools for user access to and manipulation of the hyper-network structure. This functionality is captured by the run-time layer. Due to the range of possible tools for providing this functionality, the Dexter model describes only a bare bones model for the mechanism of presenting hyperlinked content to the user for viewing and editing. The presentation mechanism captures the essentials of the dynamic, interactional aspects of HLBSs, but does not attempt to cover user interaction with the system. An important aspect of the Dexter model is the interface between the storage layer and the run-time layer. This is accomplished using the notion of presentation specifications. Such specifications are the mechanism by which information about how a component/network is to be presented to the user can be encoded into the hyper-network (at the storage layer). The way a component is presented is a function of the specific presentation tool and of properties of the component itself, including the link used to access it.

The Dexter model has spawned much research into HLBSs modelling and has become a reference against which many HLBSs are compared. The formalisation of the model has
helped it to be better understood.

However, the goal of achieving a consensus on the terminology and semantics of basic HLBSs concepts has yet to be fully realised. Furthermore, the model has been criticised in several important quarters. [Leggett and Schnase, 1994] argues that the model in its original form is insufficient for supporting large-scale HLBSs. For example, the model implicitly assumes that applications making up the within-component and run-time layers are known. This monolithic view cannot be assumed to be true. The array of different presentation, link anchoring and data storage mechanisms that could potentially be employed by distributed HLBSs imply that their behaviour and computation must be explicitly modelled and should not be modelled as an integral part of HLBSs.

Furthermore, the assumption that all data server applications model their data in the same fashion as the Dexter model’s storage layer does not hold. In distributed HLBSs, often data server applications and the data structures are not known. It is unrealistic to assume that the application data of such servers can be inspected in order to model it in the Dexter storage layer.

In summary, the Dexter model’s lack of support for distributed and multiple HLBSs is seen as a major drawback. It is also argued in [Lowe and Hall, 1999] that its implicit assumption that components of HLBSs are aware of and are tightly bound to each other is not the model which can be found in many OHLBSs developed more recently.

The model has also been criticised for not clearly distinguishing the differences between mechanisms used to reference composite components and those mechanisms used to identify atomic ones. The Dexter model does not address issues of temporal media and the deployment of links in different structural contexts, however, several proposals have been made to extend the original Dexter model to incorporate these issues. Significant examples include [Hardman et al., 1993, Grønbæk and Trigg, 1994b, Hardman et al., 1994, Grønbæk and Trigg, 1996].

Three other influential models proposed during this period, A model of Abstraction of mechanisms [Garg, 1987], the Trellis hypertext model [Furuta and Stotts, 1990] and the Formal model [Lange, 1990], are now briefly described.

In [Garg, 1987], a set theoretical model of a hyper-network is presented and a set of abstractions to provide mechanisms to retrieve and input information from various perspectives and levels of detail is defined. The stated aim of the model is to, “subjugate the complexity of understanding the different abstractions that are possible on hypertext nodes and links”. The model itself is not a complete model of HLBSs but a tool to illustrate how notions of abstraction, as applied to the fields of programming languages and databases, may be applied to HLBSs.
The relevance of this model in the context of this dissertation is that it attempts to define, at a formal level, a personalisation technique, namely content-based personalisation through information filtering. It was the first attempt to formally model how hyperlinked information might be retrieved from an information structure, from different levels of detail and from different perspectives. Such an approach may be viewed as an early attempt to investigate the possibilities of personalisation of a hyper-network by its user.

The Trellis hypertext model [Furuta and Stotts, 1990] uses Petri nets to define HLBSs in terms of different levels of abstraction. The model proposes that HLBSs can be divided into: an abstract level at which components are seen as connected in abstraction of associated presentation details; a concrete level at which the characteristics of the hypertext physical display are established; and a physical level at which the layout and presentation of the hypertext network are realised on a physical display. Each level describes the representation of part or all of the hyper-network. In contrast to the HAM and Dexter, the Trellis levels do not represent system layers. Also several aspects of HLBSs are deliberately excluded from the model (e.g., browsing semantics, dynamic behaviour, characteristics of content and user interfaces).

The Trellis model may be viewed as significant in that it allows the presence of links to be conditional on user actions. This allows controls to be placed on the accessibility of hyperdocuments. Furthermore, the approach of deliberately excluding content and user interface aspects of HLBSs is similar to that of the model proposed in this dissertation. However, the model proposed in this dissertation encompasses the dynamic aspects of HLBSs and views such aspects as central to what differentiates HLBSs from other kinds of information retrieval systems.

The Formal model presented in [Lange, 1990] has many similarities with the Dexter model. For example, the model encompasses the data structuring aspects of hyperlinked units and devotes much attention to defining nodes, links and network structures. The model goes beyond the Dexter model in that it describes the structure of individual units. A significant shortcoming of this model is that it does not address issues regarding the dynamics of user interaction with the hyper-network.

Despite the variety of early models proposed, the following shortcomings can be identified:

1. Although the models cited characterise how HLBSs manifest themselves, they fail to pinpoint what aspects of HLBSs are unique to them;

2. Although the models address issues relating to the storage, retrieval and presentation of hyperlinked information, the majority of the latest generation of HLBSs take the role of clients of other, server, technologies to provide such functionality;

3. Little consideration is given to the fact that users will often have different goals, backgrounds and experience.
The motivation for the second generation of HLBSs models was therefore driven by, among others, the following factors:

1. The requirement for the integration of HLBSs with networked computing environments and in particular, distributed storage and processing of data;
2. The requirement for the integration of different data sources and presentation mechanisms for hyperlinked information;
3. The requirement for the integration of HLBSs with other computing technologies, such as spreadsheets and word processors.

In response to these criticisms researchers began to devise models of HLBSs in which it was possible to express architectural dimensions such as third part integration, distributed data, extensibility and openness. Such models have come to be known as open models of HLBSs and the systems which exhibit such functionality as OHLBSs.

### 3.3 Modelling the Integration of HLBSs

OHLBSs research is concerned with the integration of hyperlink-based functionality with orthogonal tools in the computing environment. Generally, OHLBSs are cast as a middleware component server to storage and display clients. The approach taken is to view OHLBSs as providing these clients with hyperlink functionality. Examples of OHLBSs include Sun’s Link Service [Pearl, 1989], Microcosm [Fountain et al., 1990, Davis et al., 1992b], Devise Hypermedia Framework (DHM) [Grønbæk and Trigg, 1994a, Grønbæk et al., 1994] and Chimera [Anderson et al., 1994, Anderson, 1997].

Research in this area has focused specifically on defining open-link services (protocols) and on defining an OHLBSs architecture. This section is primarily concerned with architectural issues related to OHLBSs. In the following section, an analysis is given of the notion of openness with regard to HLBSs. Following this, several architectural models are presented and the Flag [Osterbye and Wiil, 1996], a reference model for OHLBSs is described.

#### 3.3.1 An Introduction to OHLBSs Models

The need for more open HLBSs was discussed as early as 1989 in [Meyrowitz, 1989] where Meyrowitz put forward a strong case for the need to address issues of extensibility of HLBSs.

At Hypertext’89 Sun’s Link Service [Pearl, 1989], one of the first purposely designed OHLBS was presented. The system consisted of a suite of applications which interact
with a database of links. The system provided a link service to these applications through the use of a communication protocol. Other OHLBSs quickly followed, these include Intermedia [Yankelovich et al., 1988] and Microcosm [Fountain et al., 1990].

In 1992, Davis and others [Davis et al., 1992b] published a brief list of requirements for assessing the degree of openness of HLBSs.

The list can be summarised as follows:

1. The link service provided by OHLBSs should be available across the entire range of applications within the computing environment;
2. The link service of OHLBSs should work across a network of heterogeneous platforms;
3. The architecture should be such that the functionality of the system can be extended;
4. There should be no artificial distinction between the author and reader of hyperdocuments.

This list of requirements outline early notions of what was meant by an OHLBS. Following a series of workshops [Wiil and Østerbye, 1994, Wiil and Demeyer, 1996] these initial requirements were refined and although no conclusive definition of OHLBSs has been established, the definition proposed in [Davis and Lewis, 1996] is now considered to embody the consensus of opinion on the major aspects of OHLBSs.

In [Davis and Lewis, 1996], the term open denotes the possibility of importing new objects into a system. A truly open HLBS should be open with regard to:

1. Size It should be possible to import new nodes, links, anchors and other hyperlinked objects without limitations on the size of objects or maximum number;
2. Data formats The system should allow the import and use of any data format, including temporal media;
3. Applications The system should allow any application to access the link service in order to benefit from the functionality provided by the HLBS;
4. Data models The HLBS should not impose a single view of what constitutes a hyperlink data model, but should be configurable and extensible so that new data models may be incorporated. Therefore, it should be possible to inter-operate with external HLBSs and to exchange data with such systems;
5. Platforms It should be possible to implement the system on multiple distributed platforms;
6. Users The system must support multiple users and allow each user to maintain their own private view of the objects in the system.
It should be noted that at present no single OHLBS implements all the functionality described in [Davis and Lewis, 1996]. Furthermore, those that implement specific aspects often achieve this at the cost of using proprietary solutions. With such systems, integration is possible with particular systems and particular third party applications, as opposed to a more inclusive notion of systems integration.

In [Wiil, 1997], a proposal was put forward for a reference architecture for open hypermedia [Wiil and Grønbæk, 1997]. In this proposal, a detailed indication was given of the services that should be considered important to OHLBSs and should therefore be key issues when designing a reference model for such systems. The key issues identified were:

- **Integration** OHLBSs should be capable of integrating existing services, third party applications and third party data stores;

- **Hypermedia** OHLBSs should support a rich set of hyperlink-based services such as composite objects, links, anchors, presentation and reference specifications;

- **Distribution** OHLBSs and their clients (i.e., applications, databases, browsers) should be capable of running in a distributed fashion on local and wide area networks;

- **Collaboration** OHLBSs should provide support for collaborative authoring of both content and structure. Such collaboration may be asynchronous (data sharing among a group of authors engaged in individual authoring) and synchronous (simultaneous sharing and creation of contents and structure by a group of authors).

It was proposed that the provision of open hyperlink-based services described above be flexible enough such that they may accommodate clients who required different levels and types of services. In addition, the following general architectural requirements were identified:

- **Conceptual Architecture** A reference architecture should provide a high level of abstraction for dealing with concepts, architectural components and protocols;

- **Generality** A reference architecture should be general enough to be mapped on to several physical implementations of the concepts and architectural components and comprehensive enough to allow the concept and architectural components of existing OHLBSs to be mapped on to the reference architecture;

- **Distribution simplicity** A reference architecture should be simple and easy to understand. There should be few architectural components and few protocols between components;

- **Extendibility** A reference architecture should be extendible with respect to the concepts, architectural components and protocols.
3.3.2 Architectural Models of OHLBSs

OHLBS architectures can be broadly categorised into the following:

*Hyperbase management systems* (HBMS) architectures are characterised by having a separate storage model, which handles both contents and structure, and a session manager, for assisting viewers (e.g., third party presentation tools) in maintaining contents and the structure of hyper-networks, can be identified.

The hyperbase is the middleware which instantiates the hyperlink data model and acts as an intermediary between viewers and the storage layer. Examples of HBMSs include DHM [Grønbæk and Trigg, 1994a, Grønbæk et al., 1997], SP3/HB3 [Leggett and Schnase, 1994] and HyperDisco [Wiil and Leggett, 1996, Wiil and Leggett, 1997]. Figure 3.6 depicts the architecture of an HBMS.

![Figure 3.6: Typical Architecture of a HBMS](image)

*Link server architectures* consist of a link database, which is responsible for storing the hyperlink structure (as a set of links), and a session manager, which is responsible for assisting third party viewers in maintaining this structure. Link servers explicitly distinguish between structure and contents. All data storage and presentation functionality is outside the scope of the link server. The link server provides only the links and anchoring information. Examples of systems possessing a link server architecture include Microcosm [Davis et al., 1992b] and Multicard [Rizk and Sauter, 1992]. Figure 3.7 depicts a typical link server architecture.

An *open hyperbase architecture* combines the hyperbase and link server approaches. It includes a storage module and a session layer. The storage module is responsible for storing
Figure 3.7: Typical Architecture of a Link Server
the structure and is also capable of storing the contents as well. However, contents may also be the responsibility of a third party content database. The session manager is responsible for interacting with third party presentation viewers. HyperDisco [Wiil and Leggett, 1996, Wiil and Leggett, 1997] has an open hyperbase architecture.

![Typical Open Hyperbase Architecture](image)

**Figure 3.8: Typical Open Hyperbase Architecture**

**Embedded link architectures** are characterised as consisting of two modules, a storage module and a run-time module. Such architectures make no explicit distinction between contents and structure. A classic example of an embedded link HLBS is the WWW [Berners-Lee, 1991, Berners-Lee et al., 1992], with the storage module being a WWW server and the run-time module being WWW browser. Figure 3.9 depicts the architecture of the WWW.

In summary, all architectures described above exhibit a degree of openness. However, link servers and open hyperbase architectures may be considered as more open. Both hyperbase and embedded link architectures require that third party applications can understand a specific data model in terms of structure and content formats. This constraint is not imposed on link server and hyperpage architecture.

One OHLBS reference model, the Flag taxonomy model [Osterbye and Wiil, 1996], is now briefly described.
3.3.3 The Flag Taxonomy for OHLBSs

The Flag taxonomy model [Osterbye and Wiil, 1996] may be viewed as an attempt to classify and describe individual OHLBSs. The purpose of the Flag taxonomy is to provide a framework to classify and concisely describe individual HLBSs, both open and closed, in a system independent way.

The Flag taxonomy extends the terminology of the Dexter model to adequately address the issues of integration and use of third party applications by OHLBSs.

The aim of the Flag taxonomy is to clearly distinguish between the storage, run-time, structure and contents aspects of OHLBSs. The model comprises four functional modules and four protocols. Each functional module provides functionality to be used by its neighbouring functional modules through the available protocols (see Figure 3.10).
The Flag taxonomy can be used at different levels of abstraction, ranging from classifying HLBSs into broad categories to an in-depth analysis of the individual functional modules and protocols of the system.

The taxonomy relates to the Dexter model in that the data model manager corresponds to the Dexter storage layer and is responsible for storing the structure of a hyper-network. However, the explicit distinction between structure and contents and the division of contents into storage and run-time aspects rearranges some Dexter concepts. Furthermore, the Dexter run-time layer distinguishes between concepts of instantiation (i.e., the run-time presentation of a component) and session. An instantiation consists of three parts: the component, a sequence of link markers and a mapping from link markers to anchors. In the taxonomy, this mapping belongs to the session manager module. The component and link markers are managed by the viewer module and the session is managed by the session manager module.

An important issue made explicit by the Flag taxonomy is that instantiations can be manipulated outside the structural part of HLBSs. This issue exposes the problems of integration and use of third party viewers. Figure 3.11 depicts different ways in which the Flag taxonomy views the kinds of OHLBSs discussed in section 3.3.2.

![Figure 3.11: Examples of Applying the Flag Taxonomy to HLBSs](image)

In summary, OHLBSs research aims to address issues of extendibility of HLBSs and their integration with third party applications. Such research is primarily concerned with the integration of hyperlink-based functionality with orthogonal tools in the computing environment. OHLBSs are implemented as a middleware component server to storage and display clients. The approach taken is to view OHLBSs as providing these clients with hyperlink functionality. Specific research in this area has focused on defining OHLBS models, open-link services and implementing OHLBSs.
In Section 3.4, hyperlink-based models that address user interaction issues are described. These issues include tailoring interaction, managing information in large hyper-networks, cognitive overload and disorientation.

### 3.4 Modelling User Interaction with HLBSs

The interaction process which users have with HLBSs is, in a natural sense, user controlled. Users traverse links of their choice at their own pace. However, in practice a user is restricted to the paths through the hyper-network which have been defined by the author. Users may wander through a hyper-network, but only on the paths set out by the author. This situation has led to the observation that merely providing hyperlink-based functionality to users is not enough for effective information retrieval in many cases [Brusilovsky, 1996a].

Users of HLBSs may differ in their goals, computing experience, background and knowledge of the subject covered by the HLBS. Furthermore, such differences will change over time. Many HLBSs have been criticised for providing the same traversal possibilities to all users, regardless of their goals and history. One approach to addressing this issue is to design HLBSs in which the user/system may interfere with the interaction process on an individual basis. Such interference involves designing strategies and technologies to enable the redesigning of a HLBS’s hyper-network, so that what is presented is more appropriate to each user’s information seeking goals.

PA-HLBSs is an area of research that aims to address these issues by designing models, strategies and technologies to personalise and adapt the interaction process. Nearly all research into PA-HLBSs has concentrated on implementations [Brusilovsky, 1996b]. However, in the following section, several approaches to modelling user interaction with PA-HLBSs are described.

#### 3.4.1 HLBSs Modelling using Natural Language Generation

*Natural Language Generation* (NLG) systems aim to dynamically generate a natural language text from an underlying representation of knowledge [Milosavljevic, 1996, Milosavljevic and Oberlander, 1998b]. Broadly, the approach taken is to formulate texts that satisfy a “top level” communicative goal by evaluating a set of sub-goals. To achieve this, a de-compositional process iterates over a sequence of one or more sub-goals until the resulting goals can each be realised by means of a natural language utterance.

NLG systems typically comprise two components: a *text planner* and a *surface realisation* component.
The text planner is responsible for the preparation of a discourse plan using facts from a knowledge base. This involves making decisions based on a set of discourse goals. The text planner must decide the relevance of content in a particular situation and then compose that content into a coherent text that is consistent with the current model of the user as defined by the system.

The discourse plan is realised as natural language utterances by a surface realisation component. This component makes use of knowledge of the language’s grammar and lexicon to produce well-formed utterances conveying the required content.

HLBSs have been developed using NLG techniques [Milosavljevic and Oberlander, 1998b] to tailor hyperlink-based interaction by giving users the freedom to perform high-level discourse planning. A key characteristic of these systems is that the nodes of the hyper-network are dynamically created at run-time when the user requests them. The knowledge base of such systems comprises information about concepts in the hyper-network. The NLG component of such systems is then responsible for selecting which elements of the knowledge base are important for creating the required hyperdocument.

The surface realisation component of such systems is also often responsible for the presentation of content and links. Links represent follow-up questions which the user can ask and are generally concepts (or other entities) that can be described by the system. In operation the user can effectively perform high-level discourse planning, driving the system by selecting links. Each link indicates a new discourse goal to the system. Figure 3.12 shows the traditional architecture of NLG systems and a HLBSs incorporating NLG functionality.

In [Milosavljevic et al., 1996] PEBA-II, a NLG-based HLBS that produces descriptions and comparisons of entities represented in a taxonomic knowledge base is presented. The PEBA-II system has discourse goals to describe a single entity or to compare two entities. In realising these goals, the system makes use of a user-type, user model, and a discourse history. At the beginning of each interaction with the system, the user may classify him/herself as either a naive or an expert user. This choice results in the system taking different views of the structure and content of the knowledge base. The user can also enter details into a more specific user-model which is used to make inferences about his/her specific knowledge and to draw comparisons with similar or familiar entities. The discourse history is used to tailor the output to take account of the previous discourse.

In [Milosavljevic and Oberlander, 1998a] an approach to adaptive hyperlink-based interaction is presented that uses NLG techniques to produce tailored hyperdocuments for a range of electronic catalogue systems. The focus of the approach is to generate dynamic hyperdocuments, which may then be searched and their contents tailored to an individual’s needs and backgrounds. Furthermore, these systems incorporate a range of techniques for comparing catalogue items to allow more effective searching.
Traditional Natural Language Generation

Figure 3.12: System Architecture of HLBSs Incorporating NLG Functionality
3.4.2 A Dexter-Based Reference Model for Adaptive Hypermedia

In [de Bra et al., 1999], a Dexter-based Reference Model (AHAM) is proposed. This model aims to encompass many of the adaptive features supported by adaptive HLBSs to date (see [Brusilovsky, 1996b] for further details). The model uses the terminology of the Dexter model and a set of formalisms to define several adaptive techniques.

In the model, adaptation is based on a domain model, a user model and a teaching model 3 which consists of pedagogical rules. A simple formalism is provided which illustrates how designers can write pedagogical rules about concepts in such a way that they may be applied automatically when these concepts are retrieved for presentation.

In the AHAM, HLBSs are viewed as being comprised of four parts:

- A domain model used to describe how the information in the hyper-network is structured. This model is described at the conceptual level and at the level of information fragments and pages;
- A user model used to describe the user knowledge of the concepts of a particular application domain;
- A teaching model describes pedagogical rules that indicate under which circumstances it becomes desirable or undesirable for the user to be guided towards certain parts of the hyper-network;
- An adaptive engine that is described as a software environment used for adapting content and links. The engine is envisaged to offer a library of functions for constructing information pages from fragments based on elements from the domain model, user model and teaching model.

In summary, the AHAM aims to illustrate how adaptive techniques may be implemented through the use of a set of simple formalisms of these techniques. However, in the model no distinction is made between the processes of P&A and no semantics are presented to formally define their meaning. Furthermore, the adaptive engine which is said to drive the adaptive process is only briefly described in terms of the functionality it should possess. No detailed description is given of its operation.

3.4.3 The Standard Reference Model

Intelligent Multimedia Presentation Systems (IMPSs) are concerned with the dynamic creation of multimedia presentations optimally adapted towards the needs of a user. 3

3The model uses educational terminology but its authors argue that this should not preclude it from other types of applications.
Broadly, IMPSs allow the definition of what is presented to the user (i.e., the content), how it is presented (i.e., the spatial layout) and when it is presented (i.e., temporal layout). The Standard Reference Model (SRM) [Ruggieri et al., 1996, Bordegoni et al., 1997] specifies the decomposition of an IMPS into well-defined layers. HLBSs based on the SRM may be designed to be adaptive.

The SRM divides dynamic presentation generation into two areas: generation process and knowledge server. The generation process performs the run-time generation of a presentation based on a user’s interaction, the history of the presentation and information provided by a knowledge base server. The knowledge server stores and provides long-term instructions and information that apply to multiple presentations. As such, the generation process is the active component of the SRM and the knowledge server is the stable component.

In summary, SRM defines a general processing model for dynamic presentation, generation and adaptation. The SRM can be used as the basis for comparing different systems that adapt to the user. It can also be used in guiding the development of such systems.

3.4.4 SMIL: A Personalisable Multimedia Integration Language

The Synchronised Multimedia Integration Language (SMIL) has recently been approved by the W3C working group on synchronised multimedia (SYMM). [Hoschka, 1998] SMIL is a language for describing interactive multimedia, distributed on the WWW. SMIL is an XML-based language\(^4\) that can be used to express adaptive content selection and synchronisation relationships among media elements. Using SMIL, an author can describe the temporal behaviour of a multimedia presentation, associate links with media objects and describe the layout of presentations. SMIL is defined as a set of mark-up modules which define its semantics and syntax.

During the development of the SMIL, the issue of selectability of content is addressed by the inclusion of a SWITCH construct, which is a basic selection primitive. A SWITCH allows a series of alternatives to be specified for a particular piece of content, one of which is selected by the run-time environment for presentation. The SWITCH provides a conventional branching structure that allows alternatives to be defined at authoring time.

An example of how a SWITCH might be used to control the alternative that could accompany a segment of video in a presentation would be:

\(^4\)XML, a simple dialect of SGML, is a meta-language, i.e., it enables other mark-up languages to be defined [Bray et al., 1998].
This pseudo SMIL code, specifies that a video (titlevideo.mpg) is played in parallel with one of: French audio, English audio, French text, or English text. SMIL does not specify the selection mechanism, only a way of specifying the alternatives.

SMIL also supports a partial mechanism for controlling adaptive behaviour called the system test attribute. The system test attributes consist of a set of pre-defined (primarily system-related) attributes that describe dynamic aspects of the environment which can then be tested at run-time. For example:

```
<text src="titlesong.wav" play-sound="true" ... />
```

would cause the object ‘titlesong.wav’ to be rendered (played) if the condition, play-sound, evaluates to true.

In summary, SMIL is an XML-based language with constructs that, when implemented, provide a degree of adaptive behaviour. The system test attribute mechanism is a significant extension over the SWITCH because of the way that it decouples the authoring and playback associations among a set of alternatives. SMIL is a good example of a recent raft of new languages aimed at extending the functionality of the WWW (e.g., ActiveX [Chappell and Linthicum, 1997], Cold Fusion [Danesh and Kristin, 1999, Forta, 1999]).

So far, this chapter has described and analysed many of the issues concerned with the modelling of PA-HLBSs. The issues addressed reflect important modelling trends and specific characteristics that may be viewed as important in the context of this dissertation. The analysis provided aimed to promote an understanding of the scope of hyperlink-based models and their form. In Chapters 4 to 6, an abstract model of personalisable, adaptive hyperlink-based interaction is proposed in three steps. The architectural assumptions and technical approach that act as a backdrop against which the contributions of the model are best understood is now described in section 3.5.
3.5 An Introduction to the Model for Personalisable, Adaptive Hyperlink-Based Interaction

This section describes the architectural assumptions made and technical approach taken in modelling personalisable, adaptive hyperlink-based interaction. In chapter 4, a model of core hyperlink-based functionality referred to as the H-region is defined. The H-region models non-personalisable, non-adaptive hyperlink-based interaction. In chapter 5, an abstract model of user-initiated tailoring is defined by adding to the H-region functionality comprising what is referred to as the P-region. The P-region models personalisable, non-adaptive hyperlink-based interaction. In chapter 6, an architecture to model adaptivity, the A-region, is proposed. The A-region details how system-initiated tailoring, namely adaptivity, may be modelled as a coherent and consistent future extension of the H and P-regions. The A-region therefore enables the modelling of personalisable, adaptive hyperlink-based interaction.

3.5.1 Architectural Assumptions

A general, open architecture for HLBSs of the kind depicted in Figure 3.13 as a simplified data flow diagram is assumed\(^5\). The model contributed in this dissertation proposes one view of the interior of the shaded oval in Figure 3.13.

![Figure 3.13: A General, Open Architecture for HLBSs](image)

As depicted in Figure 3.13, it is assumed that a core of hyperlink functionality is a client technology loosely-coupled to (at least) a user-interface server (UIS) and a database server (DBS).

\(^5\)The notational conventions should be familiar from classical structural analysis (see, e.g., [McDermid, 1991], p. 17/15), and can be characterised as follows. A square denotes an external entity, i.e., a producer or consumer of data that lies beyond the boundary of the function being described. An oval denotes a sub-function of the function being described. An arrow denotes the flow of the data whose name labels the arrow. In diagrams to follow, two parallel lines denote a data store into which data flows and where it is left waiting until it flows out again.
The classical example of a UIS is a WWW browser. Among other functions, browsers broker requests and have rendering capabilities. Popular browsers include Netscape Communicator [Beatty, 1997] and Microsoft Internet Explorer [Grauer, 1998], both of which instantiate a Windows/Icons/Menus/Pointer (WIMP) model. Browsers can render formal texts (e.g., rendering expressions authored in HTML). Examples of DBSs are Database Management Systems (DBMSs) which support client server architectures (e.g., Oracle [Hipsley, 1996], Informix [Taylor, 1995]).

Broadly, the dynamics associated with Figure 3.13 are as follows. The UISs capture requests for desired hyperpages. The UISs channel requests for hyperpages into the HLBS proper. If a request is for a hyperpage which resides in a remote HLBS, then the core of hyperlink functionality interacts with it to obtain the requested hyperpage in the form of a rendering expression that the core can pass back for the UISs to render. If the request is for a local hyperpage (e.g., one which is known to the core) then the latter responds by composing a rendering expression that can be rendered by UISs, possibly after querying one or more DBSs to fetch some or all of the content specified for the requested hyperpage.

It is assumed that a HLBS may rely on another (possibly remote) HLBS in the role of a server (as the WWW does, for example). In this case, it is assumed that the client HLBS plays no active role, it merely hands over the request received from the UIS to the server HLBS and hands over the response provided by the latter back to the former for it to be rendered.

Note that the main contributions of the model are not invalidated if this assumption is falsified, insofar as, strictly speaking, they are equally applicable to tightly-coupled, closed HLBSs. However, the assumption makes it easier to stress the importance of characterising the emergent properties of HLBSs and to argue that the contributions do not clash with the trend towards open architecture, but rather embrace it.

Implicit in Figure 3.13 is the assumption that P&A actions in HLBSs should not, and need not, be compounded with P&A actions that might be provided by user-interface and database components in HLBS architectures. The shaded oval in Figure 3.13 is responsible for what users experience as hyperlink-based information retrieval. Notwithstanding the fact that users may well want to personalise database and user-interface features, clearly it can be strongly argued that whatever is in scope for P&A actions in HLBSs resides in the shaded oval.

To model adaptive, personalisable hyperlink-based interaction, a model is proposed in which the shaded oval in Figure 3.13 is partitioned into three regions. Non-adaptive, non-personalisable HLBSs are modelled by the functions provided by the H-region. Personalisable HLBSs require the addition to the H-region of the functions provided by the P-region. Adaptive HLBSs require the addition to the P- and H-regions of the functions provided by the A-region.
3.5.2 Technical Approach

Technically, the approach taken is to formally model a core of hyperlink-based functionality as a *composer from specifications*, i.e., what the designer of a hyperpage designs is not a hyperpage, but rather a specification of how to build the hyperpage upon request. Hyperpages are modelled as formal specifications and a formal language is defined for this purpose.

The semantics of hyperpage specifications are given with reference to a formal abstract machine whose operation and instruction set is specified. The abstract machine is used to illustrate the execution of hyperpage specifications, thereby yielding renderable hyperpages.

Induced from the formal definition of hyperpage specifications is a set of annotation possibilities and P&A actions. These enable all design decisions realised as hyperpage specifications (and recursively their component parts) to be revised.

Personalisation is modelled as the user-initiated process of annotating and rewriting a hyperpage specification into a version thereof that is associated with the user who took that action. It follows that the hyperpages users see rendered may, if they wish, reflect their preferences, shaped by their information goals and their histories.

When personalisation functionality is layered over the core, a designer can annotate a hyperpage in preparation for differences in users’ goals and histories. A user can request to personalise not only such annotations, but the hyperpage specifications as well. Personalisation requests allow users to specify which hyperpages are to be personalised and how they should be transformed.

Hyperpage annotations and personalisation requests are modelled as formal specifications and a formal language has been defined for this purpose. Set-theoretic and relational algebraic expressions are used to represent the semantics of personalisation requests.

Adaptivity is viewed as the process of allowing the system to take the initiative in personalisation actions, in the light of the system’s inference of a user’s information goals and history.

When adaptivity functionality is layered over the core and personalisation layers, both users and designers can define strategies as to when the system should take the initiative and actively tailor the interaction to a user in the light of that user’s information goals and history of use.

In summary, the model is an abstract model, as many steps removed from concrete implementations as necessary to allow a systematic, exhaustive investigation of P&A issues in HLBSs. The model is an open model, insofar as HLBSs are viewed as clients of a variety
of servers, and in particular of data and user-interface servers. Personalisation involves a transfer of ownership of the process of interaction with a hyperdocument from designers to users. To ensure that the set of personalisation actions is consistent, its elements are induced from the formal definition of the hyperdocuments they act upon. All design decisions are, in principle, within the scope of personalisation actions.

This chapter has described and analysed many of the issues concerned with the modelling of PA-HLBSs. The analysis provided aimed to promote an understanding of the scope of hyperlink-based models and their form. This chapter has also detailed the architectural assumptions and technical approach that act as a backdrop against which the contributions of the abstract model of personalisable, adaptive hyperlink-based interaction proposed in this dissertation are best understood.

In Chapter 4, a group of functions is defined that model a core of hyperlink functionality, referred to as the H-region. The H-region functions model non-personalisable, non-adaptive hyperlink-based interaction.
Chapter 4
The H-Region: Modelling Non-Personalisable, Non-Adaptive Hyperlink-Based Interaction

This chapter defines a group of functions that model a core of hyperlink-based functionality referred to as the H-region. The functions provided in the H-region model non-personalisable, non-adaptive hyperlink-based interaction.

4.1 The H-Region: An Introduction

Within the H-region users can only request for hyperpages to be rendered. The decisions that the designers of a hyperdocument have made with respect to content, navigation and rendering cannot therefore be overridden.

Hyperpages are viewed as specifications of where to obtain content and how to render it. Upon a request arriving from a UIS, a composition function parses the hyperpage specification into a series of actions that, when executed, convert the specification into renderable text that is sent to the UIS as the response to the original request.

The semantics of a hyperpage specification are formalised as a program which, when interpreted, typically fetches content from a DBS, composes the content into a renderable text (making use of template variables as a binding mechanism) and finally responds to the original request with renderable text.

In section 4.2, the conceptual framework underlying the view of hyperpages sketched above is described. In section 4.3, descriptions of the dynamics of the H-region are provided. In section 4.4, the formal elements of the H-region are specified. The remainder of the chapter is then dedicated to describing the opportunities afforded by the H-region.
4.2 The H-Region: A Conceptual Framework

This section presents the conceptual framework underlying the H-region using a bottom-up, constructive approach in which primitive notions are presented before those derived from them.

A content specification (C-spec) defines content which is to appear in a hyperpage. A C-spec takes the form of data values (e.g., numbers, text, etc.) or more generally, requests to DBSs (i.e., query expressions which DBSs can evaluate into data values which are served back). A C-spec may be as simple as a number or a string and as complex as a sequence of complex queries which are to be sent to a variety of DBSs, possibly in many different query languages, and using many different client-server protocols.

A C-spec may be associated with a set of template variables. Conceptually, a template variable is a place holder for the content denoted by the C-spec it is associated with and which becomes available after the C-spec is evaluated. The purpose of this will be made clearer shortly.

A rendering specification (R-spec) defines how content is to be rendered by a UIS. Every R-spec is paired with one, and only one, C-spec. An R-spec takes the form of formal text in a language which the intended UIS can render, except that this renderable text may be interspersed with template variables. Template variables act as place holders for content (which may be retrieved on the fly via a DBS) as defined by the C-spec with which the R-spec is paired. After a C-spec is evaluated, the retrieved content replaces the associated template variable in the R-spec. The overall result is a renderable text (e.g., HTML).

A chunk is a pair, the first element of which is a C-spec and the second is an R-spec. A chunk may be associated with two sets of hyperpage identifiers. The first set is referred to as the set of entry points to the chunk, the second as the set of its exit points.

An entry point enables the hyperpage where the chunk occurs to be referenced in a request. If a chunk has many entry points they are construed as aliases of one another. An exit point enables a chunk to establish a navigable link to the hyperpage denoted by the exit point. In WWW parlance, an entry point can be thought of as a URL (Uniform Resource Locator) or as an anchor within a hyperpage, and an exit point as a link (e.g., an HREF tag in HTML).

A chunk is best thought of as a building block in the design of a hyperpage. Chunks are to hyperpages as atoms are to molecules.

A hyperpage is a sequence of chunks. A hyper-library is a store of hyperpages.

---

1It is assumed that this could, in turn, take the form of a reference to a local data file (containing, e.g., images, sounds, etc.). If this is the case, it is assumed that the UIS knows how to de-reference the name and get hold of the file content for rendering. Under this assumption, values and references to files containing values are considered to be conceptually the same from the point of view of the model.
the purposes of the model, it is assumed that the implementation of the hyper-library provides the functionality of a modern database system, even a DBS on par with, and perhaps indistinguishable from, those where content is sourced from. In particular, it is assumed that there exist mechanisms for scalable retrieval, associative querying, security of access, concurrency, versioning, and transaction control.

Note, there is no need to postulate that hyperpages are further structured to form a hyperdocument, i.e., a collection of hyperpages whose formal properties enable certain navigational operations to be performed over it. For example, with reference to an unordered collection and one of its members, one can only request another member. If, however, the collection is known to be totally ordered, then requests for the next and previous member are meaningful. Analogously, if the collection is a tree, then requests for the root, or for the children of a non-leaf node, among others, are meaningful. Clearly, this issue of designer-imposed structure on collections of hyperpages is a very important one but it is also orthogonal to the model and therefore one that is not further addressed in this research.

A designer is an author of hyperpages. In this role, the designer is required to decide on the sequence of chunks that make up the hyperpage. This, in turn, involves defining for each chunk its entry points, if any, its exit points, if any, and its C-spec and R-spec. The specification of entry and exit points binds the hyperpage being designed with other hyperpages, not necessarily local ones. Analogously, the C- and R-specs may bind the hyperpage to different DBSs and different UISs. It follows that a designer is engaged in two kinds of decision: on the one hand the designer must decide on how the hyperpage will be accessible and what other hyperpages it will provide access to; on the other hand the designer must decide on what the hyperpage will contain and how this content will be presented.

A user requests that hyperpages be rendered by interaction with a UIS. Thus, the user might click on a BACK button, or click on a link or type a hyperpage identifier into a GO-TO window. The observable result of processing such a request is the display of the hyperpage as specified by its designers.

This is the conceptual framework underlying the H-region. Subsection 4.4 provides formal definitions of the concepts discussed.

4.3 The H-Region: Dynamics of the H-Region

The dynamic behaviour of a HLBS instantiating the H-region is now informally described. Figure 4.1 is a simplified data flow diagram\(^2\) which expands the shaded oval in Figure 3.13.

\(^2\)The notational conventions drawn from classical structural analysis (see, e.g., [McDermid, 1991], p. 17/15), can be characterised as follows. A square denotes an \textit{external entity}, i.e., a producer or consumer.
Figure 4.2 is a simplified data flow diagram which expands the perform retrieval function in figure 4.1.

Figures 4.1 and 4.2 illustrate that designers specify hyperpages and store them in the hyper-library. When a well-formed user request comes from the UIS, the H-region resolves it into one of the following:

- A reference to a logged hyperpage (e.g., the request is 'BACK' which resolves into the hyperpage that is logged immediately before the current hyperpage).

Notational conventions are the same as those of Figure 4.1.
In this case, the H-region can fetch the corresponding rendering expression from the cache and return it to the UIS for rendering. If the request had included a refresh request, then it would have been treated as if the hyperpage had never been logged;

- A reference to a hyperpage that resides in a remote HLBS.

In this case, the H-region forwards the request to the latter and upon return of the corresponding rendering expression forwards it to the UIS for rendering;

- A reference to a hyperpage that resides in the hyper-library of the HLBS proper.

In this case, the H-region fetches the hyperpage from the hyperlibrary and proceeds to parse and compose it.

In the case of the H-region, a well-formed request always denotes a hyperpage. The UIS does not effect any transformation in the request and simply forwards it to the H-region. There is no opportunity for P&A and hence, no possibility of ownership transfer.

The composition function parses the hyperpage specification fetched from the hyper-library into a set of abstract machine instructions that, when interpreted, assemble the renderable text. This interpretation process may involve fetching content specified as a query to a remote DBS. It is the returned content that is woven into the renderable text, thereby characterising the dynamic nature of hyperpages in the model.

The specification language for hyperpages (and, recursively, for any component parts) is defined by a formal grammar given in subsection 4.4.2. The composition function in Figure 4.1 includes a parser for this language whose output consists of commands on where and how to retrieve the contents of the hyperpage and how to compose the corresponding rendering expression. The composition function also implements the operational semantics of the language as an interpreter for the commands output by the parser. When the interpreter finishes executing the program the finalised rendering expression is returned to the UIS, which requested the hyperpage, for rendering.

Note that the overall operation of the composition function may well be governed by broad behavioural parameters (e.g., default actions regarding the implementation of caching and proxy server technologies).

This subsection has described the dynamic behaviour of a HLBS instantiating the H-region. The H-region is modelled by a hierarchy of functions whose signatures, as well as the types used in them, are given in Figure 4.3. Appendix B details a complete procedural design of the H-region.
4.4 The H-Region: Formal Elements of the H-Region

The following section defines the formal elements of the H-region. These include an EBNF grammar for hyperpage specifications and the formal semantics of hyperpage specifications given with reference to a simple abstract machine. Section 4.5.1 formally defines this abstract machine.

4.4.1 An Introduction to Context Free Grammars

In the proposed model a formal language, defined by a context-free grammar, is used to define the syntax of hyperpage specifications. To aid an understanding of this language this subsection contains a brief recapitulation of basic definitions from formal language theory.

A language can be viewed as a (usually infinite) set of sentences or strings. Such strings are composed of symbols of some alphabet. However, not all combinations of symbols are well-formed strings. Thus, when defining a new, or describing an existing language, the specification should contain only well-formed strings. A number of formalisms have been suggested to facilitate such systematic descriptions of languages, most notably the formalism of context free grammars (CFGs) [Wood, 1987].

Formally a context-free grammar is a 4-tuple \( (S, \mathcal{P}, \mathcal{N}, \mathcal{T}) \), where \( \mathcal{N} \) and \( \mathcal{T} \) are finite, disjoint sets of identifiers called nonterminal and terminal alphabets respectively. \( \mathcal{P} \) is a finite subset of \( \mathcal{N} \times (\mathcal{N} \cup \mathcal{T})^* \) where \( (\mathcal{N} \cup \mathcal{T})^* \) denotes the set of all strings (sequences) of terminals and nonterminals. \( S \) is a nonterminal symbol called the start or sentence symbol, \( S \in \mathcal{N}. \) Traditionally the empty string is denoted by the symbol \( \varepsilon. \) Elements of the relation \( \mathcal{P} \) are
usually written in the form:

\[ A \rightarrow B_1 \ldots B_n \text{ where } n > 0 \]
\[ A \rightarrow \varepsilon \text{ when } n = 0 \]

Each such element is called a *production rule*. The general form of a production rule is

\[ A \rightarrow B_1, B_2, B_3 \ldots B_n \]

This defines the entity \( A \) as being made up of a string of simpler entities \( B_1, B_2, B_3 \ldots B_n \). These may be actual tokens appearing in the program, or they may themselves be the subject of a definition elsewhere. Informally, anywhere the entity \( A \) appears it may be replaced by the string \( B_1, B_2, B_3 \ldots B_n \). If it is well-formed, eventually a string will remain that contains nothing that can be expanded further, such a string is called a *sentence*. In the context of programming languages, syntactically correct programs are sentences derived using the context-free grammar defining the syntax of the programming language. Therefore a sentence of a context-free grammar may be formally defined as follows:

Let \( G \) denote a grammar that is a 4-tuple \( \langle S, P, N, T \rangle \) where \( S \) is the start symbol, \( P \) is a set of productions, \( N \) is a set of nonterminal symbols, and \( T \) set of terminal symbols. Finally let \( V \) denote the alphabet, \( N \cup T \), then;

A sentence is a string of symbols in \( T \) derived from \( S \) using one or more applications of productions in \( P \).

Each production has the form \( X \rightarrow v \), where \( v \) is an arbitrary string of symbols in \( V \) and \( X \) is a single nonterminal.

Using this definition the language, \( L(G) \) defined by a grammar, \( G \), is the set of sentences derivable using \( G \).

The classical approach to specifying the syntax of a programming language defined by context-free grammar is known as Enhanced Backus-Naur form (EBNF) [Wood, 1987], a particular notation, well suited to representing context free grammars. This notation is used in the following section to specify the syntax for hyperpage specifications.
4.4.2 A Formal Grammar for the Specification of Hyperpages

An EBNF grammar\textsuperscript{4} for user requests to the H-region is given in Figure 4.4. Note that all requests are for traversal so that data can be fetched, i.e., they all resolve into hyperpage references, either absolute or relative.

Figure 4.4 is an example only. It could be extended in many ways. Its purpose is to stress the crucial factor in terms of describing the functionality of the H-region: no P&A actions are supported.

\begin{verbatim}
request ::= data-retrieval-request
data-retrieval-request ::= go-to hyperpage-reference
hyperpage-reference ::= this
| again
| previous
| next
| HYPERPAGE-ID
\end{verbatim}

Figure 4.4: An EBNF Grammar of User Requests to the H-Region

The EBNF grammar in Figure 4.5 defines the formal syntax of hyperpage specifications. An example of a hyperpage specification, i.e., a well-formed string in the language defined by the grammar in Figure 4.5, is given in Figure 4.6.

Some, more or less inessential, choices taken in the grammar in figure 4.5 are as follows. Keywords that introduce component parts are delimited by a pair of curly brackets. For example each chunk is headed by the keyword `chunk` and has its scope delimited by the pair `{}).

In the example in Figure 4.6, string values are delimited by single-quote pairs (e.g., `'hello'`). Other values (e.g., integers, characters, etc.) are assumed to be representable as in modern programming languages. Representations of values of primitive types are elements of the terminal class `VALUE`. Similarly, the terminal class `IDENTIFIER` from which names for template-variables are drawn is assumed to be similar to the class of variable names in modern programming languages, although here there is a preference for naming them with a single `UPPERCASE-LETTER`, possibly followed by an integer, drawn from the end of the Latin alphabet (e.g., `X, Y1, etc.`).

The remaining classes of non-terminals are `UIS-STRING` and `DBS-STRING`. Their elements

\textsuperscript{4}Notational conventions are as follows. Nonterminals are shown in \textit{sans-serif font}, except the root symbol which is shown in \textit{slanted sans-serif font}. Terminals are shown in \textit{typewriter font} unless they form a class, in which case the said class is shown in \textbf{UPPERCASE-LETTERS}. Possibly empty sequences are denoted by an `*` appended to the category from which the terms of the sequence are drawn, and nonempty ones by `+`. An `*` (or `+`) may be annotated with a separator character (e.g., `(, )`), indicating that all terms but the last are followed by a terminal which separates the terms (`, ` in the example just given). The character `%` is used to denote the start of a comment on the line it occurs.
\[
\text{hyperpage} ::= \text{page} \{
\text{chunk}^* \\
\}
\]
\[
\text{chunk} ::= \text{chunk} \{
\text{entry-point}^* \\
\text{C-spec} \\
\text{R-spec} \\
\text{exit-point}^* \\
\}
\]
\[
\text{entry-point} ::= \text{entry} \{
\text{UIS-STRING}^+ \\
\}
\]
\[
\text{C-spec} ::= \text{content} \{
\text{content-assignment}^*, \\
\}
\]
\[
\text{R-spec} ::= \text{rendering} \{
\text{rendering-element}^* \\
\}
\]
\[
\text{exit-point} ::= \text{exit} \{
\text{UIS-STRING}^+ \\
\}
\]
\[
\text{rendering-element} ::= \text{template-variable} \mid \text{UIS-STRING}
\]
\[
\text{content-assignment} ::= \text{template-variable} ::= \text{content-expression}
\]
\[
\text{template-variable} ::= \text{IDENTIFIER}
\]
\[
\text{content-expression} ::= \text{VALUE} \mid \text{DBS-STRING}
\]

Figure 4.5: An EBNF Grammar of Hyperpages
are elements of the language which, respectively, the UIS can render and which the DBS can evaluate. In the example above, elements of both UIS-STRING and DBS-STRING are delimited by a pair of square brackets (e.g., \[<I> <B>\], or \[SELECT * FROM welcome\]) so that they can be distinguished from the context. In the example, the assumption is that the UIS can render HTML and that the DBS can evaluate SQL queries. It may well be that these strings may be more complex, e.g., a DBS-STRING may need to indicate a protocol (e.g., ODBC) and a desired server. However, these strings have no H-region specific semantics to allow appropriate languages to be incorporated as required. Therefore their denotations are not further discussed in the model.

A greater degree of structure could be assumed of both UIS-STRING and DBS-STRING. One might make them non-terminals, e.g., HTML-string and ODBC-request. This step is not taken because it does not bear on the main purpose of the model, since such a step would only require more complex, but well-established, technology to be brought under the control of the H-region. However, this should not be taken to mean that such a step is not correct or not advisable.

A similar point arises with respect to link server technology. By an analogous argument one can explain the non-inclusion of a link server connection. Clearly, rather than defining an exit-point to be a nonempty sequence of UIS-STRING it could have been defined as a query to a link server exactly like a content-expression can be a query to a data server.

A hyperpage, of which Figure 4.6 is an example, is a specification of the content to be presented to the user and the rendering of this content. The semantics of a hyperpage is an abstract program that when interpreted, retrieves the specified content and constructs the rendering text intended by the designers.

Note that, at this level of abstraction, a hyperpage specification need not have an identifier beyond the one (e.g., a path name) which allows it to be fetched from the hyper-library. Analogously, the constituent parts of a hyperpage specification can, and will later on, be referred to by their relative position in the hyperpage specification.

The hyperpage in Figure 4.6 has two chunks, the first of which has one entry point (viz., \(<A NAME="welcome">\</A>\)) and no exit point, while the second has no entry point and one exit point (viz, \(<A HREF="http://www.gold.ac.uk/">next \</A>\)). An entry point defines an access point internal to the hyperpage. An exit point defines a hyperlink, i.e., a point at which access may be gained to the corresponding hyperpage.

Every chunk has a possibly empty C-spec, and a possibly empty, R-spec. In the first chunk, the C-spec consists of two assignments, the first assigns the value ', hello' to the template variable \(X\) and the second assigns the value 'world' to the template variable \(Y_1\).

Content assignments are very similar to standard destructive assignments in imperative
languages, hence the choice of the symbol :=. The scope of template variables is local to the chunk in which they occur.

The R-spec in the first chunk uses HTML notation to render the content of X in boldface and the content of Y1 in italic boldface. The occurrence of template variables within an R-spec signals that textual replacement operations need to be carried out to obtain a rendering expression that can be returned to the UIS.

The noteworthy feature of the second chunk is in its C-spec where a query expression (SELECT * FROM welcome) is specified for the DBS. The results returned by the DBS are assigned to the template variable X.

Using simple binding and textual replacement primitives, and assuming the ability to operate client-server protocols, one can define a formal semantics of hyperpage specifications in terms of an extremely simple state-based abstract machine. The composition function in Figure 4.1 denotes such an abstract machine: it executes the program output by the parser of hyperpage specifications such as the one in Figure 4.6. In subsection 4.5, the formal semantics of hyperpage specifications are defined with reference to a state-based abstract machine.
4.5 Formal Semantics of Hyperpage Specifications

In this section the formal semantics of hyperpage specifications is given with reference to a simple abstract machine whose instruction set is shown in Figure 4.7.

4.5.1 Hyperpage Abstract Machine

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Denotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>bind</td>
<td>IDENTIFIER' × location</td>
</tr>
<tr>
<td>evaluate</td>
<td>content-expression' × location</td>
</tr>
<tr>
<td>compose</td>
<td>rendering-expression' × location</td>
</tr>
<tr>
<td>replace</td>
<td>IDENTIFIER' × location1 × location2</td>
</tr>
<tr>
<td>append</td>
<td>location1 × location2 × location3</td>
</tr>
<tr>
<td>return</td>
<td>location</td>
</tr>
</tbody>
</table>

Figure 4.7: Abstract Machine Instructions for the Hyperpage Composition

In Figure 4.7, IDENTIFIER', content-expression', rendering-expression' are the denotations of the corresponding syntactic constructs and location_n denotes an address in the memory space of the abstract machine.

Informally, the intended meaning of these instructions is as follows:

- **bind** allocates memory space at location to the template variable whose denotation is IDENTIFIER';
- **evaluate** writes onto location the value of content-expression' (which may originate from a data server to which the abstract machine, in the role of client, submits a retrieval request);
- **compose** writes rendering-expression' onto location;
- **replace** moves the contents of location_1 onto location_2, replacing each occurrence of IDENTIFIER' in location_2 with the contents of the memory space bound to IDENTIFIER' into location_1;
- **append** moves onto location_3 the concatenation of location_1 and location_2;
- **return** passes back location, which contains the expression to be rendered, to the UIS.

Figure 4.8 shows the program which is output produced by the parser in an assembly language postulated on the basis of the abstract machine instructions. Strings in this language are elements of the type program in Figure 4.3.

In Figure 4.8, the following notational conventions are adopted. The denotation of a rendering-expression or of a content-expression is a string and a string is enclosed in single
quotes. Memory locations are denoted mnemonically by identifiers beginning with an upper-case \( L \) and followed by an integer. The mnemonic of a memory location followed by a ‘*’ denotes the address of (i.e., a pointer to) that location. Note that, by abuse of notation, strings may contain references to memory locations. The denotation of a template variable has its scope resolved by appending to it the position of the chunk where it occurs relative to the beginning of the source text.

### 4.6 Opportunities Afforded by the H-Region

The abstract, open model of hyperlink-based interaction described in this chapter is one answer to the question of what are the emergent properties of HLBSs. It qualifies as one such answer insofar as it does not model any capability that might be inherited from server technologies. This conclusion notwithstanding, it is acknowledged that a different conceptual framework would induce a different view of HLBSs and hence, a different answer to the question above. Therefore, the model does not claim to be the only possible characterisation of the emergent properties of HLBS. Furthermore, the body of HLBSs modelling work, both closed and open, provides alternative characterisations, albeit largely implicitly.

The contribution of this dissertation lies, in this respect, in providing one such characterisation and doing so explicitly, insofar as the emergent properties of the class of HLBSs defined are unambiguously set in contrast to the properties that such systems can simply inherit from other technologies. In the model, HLBSs can be seen to multiply inherit functionality from user-interface and database systems, but they exhibit specific functionality:
exactly that which is modelled by the H-region.

As suggested in chapter 1, this is equivalent to saying that the scope for P&A actions, taking this view of HLBSs, is the interaction emerging from the H-region. This emergence, in turn, is completely determined by hyperpage specifications. It follows that if one wants to personalise that interaction all that is needed, and all that can be done, is to personalise hyperpage specifications. Once a hyperpage specification is personalised by a user, the functionality of the H-region delivers personalised interaction without requiring any change or redesign, let alone re-implementation.

In summary, the abstract model of core hyperlink-based functionality contributed by this chapter, possesses the following beneficial characteristics:

1. **Data type and media type independence**: All forms of data and media may be referenced using hyperpage specifications written in the formal grammar defined for hyperpages;

2. **Content and presentation independence**: The conceptual framework of the H-region allows for the specification of independently defined and independently accessible units of contents (chunks). The presentation of these units of information is separated for their content. Any appropriate presentation language (e.g., HTML) may be incorporated seamlessly for a particular unit of information. Furthermore, several different mark-up languages may appear in the same hyperpage;

3. **Freely defined links**: The model allows for the inclusion of freely defined links between freely defined units of information. Although in the model the use of link servers is not specifically addressed, it is clear that the incorporation could be completed along the same lines as that for the inclusion of remote content;

4. **Dynamically generated hyperdocuments**: The model proposes a hyperpage specification for the dynamic generation of hyperdocuments. These hyperdocuments have no inherent structure, hierarchical or otherwise. However, nothing precludes inclusion in the model of hyperdocuments whose content is static;

5. **A core of hyperlink functionality**: The model clearly delineates the scope of P&A actions as a core of functionality which can (and indeed should) be realised as a client technology of content and presentation servers.

### 4.7 What Next?: A Model of User-Initiated Personalisable Hyperlink-Based Interaction

In chapter 5, a model of user-initiated personalisable hyperlink-based interaction, the P-region, is presented. The model illustrates how the formal characterisation of hyperpage
specifications induces a set of personalisation actions that allow a user to override all the design decisions that a hyperpage specification embodies. It is also shown that the P-region provides the functionality needed for personalisation actions without disrupting, and, in fact, relying on the functionality of the H-region remaining intact and unaltered.
Chapter 5
The P-Region: Modelling
User-Initiated Personalisable
Hyperlink-Based Interaction

Chapter 4 described an abstract, open model of non-personalisable, non-adaptive
hyperlink-based interaction. This chapter extends the results of chapter 4 by defining
a further group of functions, the P-region, so that user-initiated tailoring actions are pos-
sible.

5.1 The P-Region: An Introduction

The P-region comprises a group of functions that are non-disruptively added to the
H-region in order to model personalisable hyperlink-based interaction. Within the P-region
users can not only request a hyperpage, but can also annotate or rewrite it, thereby cre-
ating their own version of it. The decisions that the designers of that hyperpage have
made with regard to content, navigation and rendering of the hyperpage can therefore be
overridden by users and this kind of event characterises ownership transfer.

The kinds of personalisation actions modelled are based on annotating and rewriting the
hyperpage specifications detailed in chapter 4. Annotation pairs a hyperpage specification
with notes of interest to the user and, by doing so, presumes that versioning takes place.
Such notes take one of the following forms. Firstly, a note can assign user-specific values
to user-generic attributes of interest (e.g., that the level of difficulty of a given page or
component part is high, or that ‘planets’ is a keyword of relevance to a given page). Secon-
dly, a note can specify a rewriting action over the renderable text after it has been
composed by the H-region, i.e., after content has been fetched and made ready for display
(e.g., to map American into British spelling forms). Rewriting actions on hyperpage
specifications allow any part of any hyperpage to be updated.
The existence of annotations on hyperpages allows for:

1. **personalisation** of a specified hyperpage;
2. the specification of **alternatives** to a specified hyperpage;
3. the specification of **comparable hyperpages** to a specified one;
4. the recording of **information** about a hyperpage (i.e., what are the current values of attributes set by previous annotations).

Annotations are not operations on hyperpage specifications, i.e., they do not edit the latter. Rather, they give rise to a user-specific pairing with a hyperpage specification.

Rewriting also causes versioning and can be characterised simply as the editing of hyperpage specifications. If a user edits the hyperpage specification as conceived by its designers, ownership is *transferred*. If, subsequently, the same user edits that hyperpage specification again, the personalisation process is *fine-tuned*.

Section 5.2 details the conceptual framework that underlies the P-region. This section describes the new concepts which are introduced to model personalisable hyperlink-based interaction. In section 5.3, the dynamic behaviour of a HLBS instantiating the P-region is informally described. Section 5.4 defines the formal elements of the P-region. The section begins by introducing a formal grammar for the specification of user requests. Following this, a formal grammar is specified for the annotation of hyperpage specifications detailed in chapter 4. In subsection 5.4.3, a formal grammar is presented for the specification of personalisation requests. Such requests enable the designer or user to rewrite a hyperpage specification completely if required. Several examples of personalisation requests specified using the grammar are then given in subsection 5.4.4.

The second half of this chapter is dedicated to defining the formal semantics for personalisation requests. In section 5.5, the approach taken is introduced. In subsection 5.5.1, hyperpages and hyperpage annotations are modelled using the relational data model.

Broadly, this subsection defines the hyper-library as a set of relational schemas. Subsection 5.5.2 then details a set of preliminary definitions which are needed to define the semantics of personalisation requests. Among others, definitions are given of functions which specify the reconstruction of the hyper-library from a set of relations. Finally, in subsection 5.5.3, the semantics of personalisation requests is defined, in terms of relational algebra with assignments. The chapter concludes by describing opportunities afforded by the P-region.

---

1 In appendix F a description can be found of the basic concepts which underpin the relational data model. This appendix also briefly describes the relational algebra used to operate on relations.
5.2 The P-Region: A Conceptual Framework

This section presents the new concepts introduced by the P-region that form its conceptual framework.

A note is a valuation of an attribute of a hyperpage or of one of its component parts. In the model a set of attributes must be specified, but the set of attribute values need not be. In this sense, it can be said that the set of attributes is user generic and that the set of attribute values is user specific.

Given an attribute and a hyperpage structure (i.e., an entire hyperpage or a component thereof), a note is an assignment of a value to the given attribute in the scope of that structure. More generally, a note can be viewed as a construct which provides user-specific semantics to some aspect of a hyperpage which may, for example, inform the way preferences are enforced. Notes determine the current view that designers or users have on different aspects of a hyperpage (e.g., what is its subject, what level of complexity is associated with the content, what navigational alternatives are comparable in content). When notes are attached to hyperpage components then a function in the P-region can be parameterised by their values.

An annotation is a sequence of notes associated with a hyperpage.

A personalisation request is an editing command over hyperpage specifications that causes a modified version of the hyperpage to be versioned by the user who issued the personalisation request. A personalisation request specifies which hyperpages to personalise and what to transform them into. A personalisation request is therefore a request to override the original decisions of the designers of the hyperpage (and, of course, past expressions of preference by the user). Any design decision can, in principle, be overridden (i.e., a set of personalisation requests can, in principle, rewrite a hyperpage completely).

All the concepts introduced in Section 4.2 are retained without change, except that users and designers now have more actions that they can perform. A user may also request a hyperpage to be annotated or rewritten. It is also open to the designer to generate the hyperpage with annotations.

A hyper-library becomes a store of hyperpages and annotations. It is assumed that a P-region implementation (perhaps at the hyper-library) enforces a one-to-one mapping between annotations and hyperpages versioned to the user (or possibly group or users) that made the annotations. This, in turn, assumes a versioning capability, as well as concurrency and transaction control.

This is the conceptual framework underlying the P-region. Section 5.4 provides formal definitions of the concepts discussed here.
5.3 The P-Region: Dynamics of the P-Region

The additional dynamic behaviour of a HLBS instantiating the P- and H-regions is now informally defined. Figure 5.1 is a simplified data flow diagram. P-region functions do not conflict with H-region functions and, indeed, rely on their being exactly as defined in Figure 4.1, except for the perform retrieval function that now handles more cases and the request data flow and the hyper-library data store which are enriched as already discussed. This match is graphically represented by super-imposition, with the functions and flows that are identical in Figure 4.1 and Figure 5.1 drawn with thinner lines and named with a lighter typeface.

![Figure 5.1: Superimposing the P-Region onto the H-Region](image)

Figures 5.1 illustrates that the P-region provides two basic processes: the personalisation of hyperpages by annotation and rewriting and the enforcement over a renderable text of previously expressed preferences (in the form of notes on a hyperpage). In Figure 5.1, the personalise function embodies the former, while the apply preference function embodies the latter.

The personalisation process starts with a personalisation request which a user conveys to the UIS. A personalisation request specifies a scope (i.e., which hyperpages it acts upon) and the actions (which may simply be annotations) that the user wishes to effect. The
personalise function parses the request into a set of instructions that, when interpreted, retrieve from the hyper-library the hyperpages in scope, carry out the actions specified in the personalisation request over the retrieved hyperpages to generate their user-specific versions and write the latter into the hyper-library. In section 5.5, the formal semantics of personalisation requests is given.

The actions specifiable in a personalisation request are the rewriting and annotating of hyperpages and hyperpage annotations. Thus, a versioned hyperpage may be an edited version, or its pairing with an annotation or both, concomitantly or not.

One possible note is to specify a post-composition rewrite. When one such specification exists for a hyperpage, the P- and H-regions interact under the control of the perform retrieval function. Thus, when faced with a data retrieval request, the perform retrieval function queries the hyper-library as to whether the required hyperpage is annotated with post-composition rewrites. If it is not, then only H-region functionality is needed to respond to the UIS. If the required hyperpage is annotated with post-composition rewrites, then the P-region ensures that, after the composition function returns the renderable text, the rewriting is effected by the apply preference function before a response is dispatched to the UIS. Note that this rewriting may be conditional on the environment, therefore the apply preference function may need to query the environment to verify whether the state of the environment blocks, or instead forces, the rewriting.

Other notes cause changes in the behaviour that would not be manifested in the absence of the P-region. For example, given two hyperpages a note may specify that one is an alternative or is comparable to another. This allows requests for alternative and for comparable hyperpages to a given one. Finally, since annotations are effectively metadata, a user may choose to request, rather than a hyperpage, information about it.

This subsection has described the dynamic behaviour of a HLBS instantiating the P-region. The P-region is modelled by a hierarchy of functions whose signatures, as well as the types used in them, are given in Figure 5.2. Appendix C details a complete procedural design of the P-region.

5.4 The P-Region: Formal Elements of the P-Region

This section defines the formal elements of the P-region. These include EBNF grammars for the specification of user requests, personalisation requests, hyperpage annotations and information requests. In section 5.5, a recursive function that defines the formal semantics of personalisation requests is given. This function traverses an abstract syntax tree representation of a personalisation request and generates the corresponding semantic representation as a sequence of assignments and relational algebraic expressions.
<table>
<thead>
<tr>
<th>Function</th>
<th>Input Type</th>
<th>Output Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>classify-request</td>
<td>request</td>
<td>nature-of-request</td>
</tr>
<tr>
<td>perform-retrieval</td>
<td>request</td>
<td>outcome × ren-exp</td>
</tr>
<tr>
<td>classify-hyperpage-reference</td>
<td>hyperpage-reference</td>
<td>nature-of-hp-reference</td>
</tr>
<tr>
<td>fetch-remote</td>
<td>hyperpage-reference</td>
<td>outcome × ren-exp</td>
</tr>
<tr>
<td>fetch-logged</td>
<td>hyperpage-reference</td>
<td>outcome × ren-exp</td>
</tr>
<tr>
<td>convert-to-render</td>
<td>hyperpage-reference</td>
<td>outcome × ren-exp</td>
</tr>
<tr>
<td>classify-hyperpage-state</td>
<td>hyperpage-reference</td>
<td>hyperpage-state</td>
</tr>
<tr>
<td>parse-hyperpage</td>
<td>hyperpage-reference</td>
<td>H-program</td>
</tr>
<tr>
<td>compose-hyperpage</td>
<td>H-program</td>
<td>ren-exp</td>
</tr>
<tr>
<td>fetch-personalised</td>
<td>hyperpage-reference</td>
<td>hyperpage × annotation × outcome</td>
</tr>
<tr>
<td>apply-rewrites</td>
<td>annotation × ren-exp</td>
<td>ren-exp</td>
</tr>
<tr>
<td>parse-pr</td>
<td>personalisation-request</td>
<td>P-program</td>
</tr>
<tr>
<td>Interpret-pr</td>
<td>P-program</td>
<td></td>
</tr>
</tbody>
</table>

request ⊆ STRING
hyperpage-reference ⊆ STRING
ren-exp ⊆ STRING
nature-of-hp-reference ⊆ STRING
nature-of-request ⊆ STRING
hyperpage-state ⊆ STRING
outcome ≡ BOOLEAN

Figure 5.2: P-Region Functions: Signatures and Types

personalisation-request (see Figure 5.7–5.8)
hyperpage (see Figure 4.5)
H-program (see Figure 4.8)
annotation (see Figures 5.4–5.5)
P-program (see Section 5.5)
5.4.1 A Formal Grammar for the Specification of User Requests

An EBNF grammar for user requests to the P-region is given in Figure 5.3. Note that this grammar extends (i.e., has shared non-terminals with) the grammar for requests to the H-region given in Figure 4.4 and assumes non-terminals only defined in the grammar for annotations given in Figures 5.4–5.5.

| request          ::= data-retrieval-request  |
|                  | personalisation-request  |
|                  | information-request  |
| data-retrieval-request ::= go-to hyperpage-reference  |
|                      | go-to-alternative-of hyperpage-reference  |
|                      | go-to-comparable-to hyperpage-reference  |
| hyperpage-reference ::= this  |
|                      | again  |
|                      | previous  |
|                      | next  |
|                      | HYPERPAGE-ID  |
| information-request ::= show attribute* at scope with hyperpage-reference  |

Figure 5.3: An EBNF Grammar of User Requests to the P-Region

Note that now, besides requests for data to be fetched, users can also request the personalisation of hyperpage specifications and for information on hyperpage specifications to be provided. Information requests are requests that the attributes of a hyperpage, as set by annotation, be displayed for some hyperpage specification. Note also that data requests can now also be for alternative or comparable hyperpages to the hyperpage that is current.

Annotations (and notes) are formally defined in Figures 5.4–5.5. Personalisation requests are formally defined in Figures 5.7–5.8. Their formal definition, in turn, assumes annotations to be defined.
### 5.4.2 A Formal Grammar for the Specification of Hyperpage Annotations

An EBNF grammar which defines the formal syntax of hyperpage annotations is given in Figures 5.4–5.5. Recall that a hyperpage is comprised of a sequence of chunks and that each chunk is comprised, at least, of a single C-spec and a single R-spec. Furthermore, that a chunk may also have multiple entry and exit points.

\[
\begin{align*}
\text{annotation} & : = \text{annotation} \{ \\
& \quad \text{note}^* \\
& \} \\
\text{note} & : = \text{scope attribute-assignment} \\
& \quad | \text{rewrite} \\
& \quad | \text{QUERY-ON-ENVIRONMENT rewrite} \\
\text{scope} & : = \text{page} : \\
& \quad | [ \text{page-part} ] : \\
\text{page-part} & : = \text{relevant-chunk} \\
& \quad | \text{relevant-chunk ( relevant-single-part )} \\
& \quad | \text{relevant-chunk ( relevant-multi-part )} \\
\text{relevant-chunk} & : = \text{shift}, \text{chunk} \\
\text{relevant-single-part} & : = \text{C-spec} \\
& \quad | \text{R-spec} \\
\text{relevant-multi-part} & : = \text{shift}, \text{entry-point} \\
& \quad | \text{shift}, \text{exit-point} \\
\text{shift} & : = \text{any} \\
& \quad | \text{signed-integer} \\
\text{signed-integer} & : = \text{INTEGER} \\
& \quad | \text{sign INTEGER} \\
\text{sign} & : = + \\
& \quad | -
\end{align*}
\]

Figure 5.4: An EBNF Grammar of Hyperpage Annotations (I)
attribute-assignment ::= attribute := attribute_value

attribute ::= description | keyword | level
| see-as-well | see-instead
| wherefrom | whereto
| status

attribute_value ::= VALUE
| entry-point*
| exit-point*

rewrite ::= scope from -> to

from ::= REGULAR EXPRESSION

to ::= REGULAR EXPRESSION

Figure 5.5: An EBNF Grammar of Hyperpage Annotations (II)
An annotation is a nonempty sequence of notes. A note is either an attribute assignment or a (post-composition) rewriting specification, where the rewriting can be unconditional or conditional on the state of the environment.

If a note is a rewriting specification, then it must define what strings to rewrite and what strings to rewrite the latter into. In Figures 5.4–5.5, the non-terminals from-regular-expression and to-regular-expression are regular expressions as defined in section 5.5.1.

If a note is an attribute assignment, then it must specify the scope of the assignment. This scope is either the entire page, or one of its component parts. In the latter case, the exact component part is specified by a data structure that completely determines the component part in the page using relative displacements.

The set of attributes to which values can be assigned is fixed for all users, since only then can their semantics be defined and made use of in, e.g., data requests. The set of values which attributes can take might be fixed for all users, but it need not. A fixed set of attributes is modelled for hyperpages, primarily to illustrate their use, however, there is nothing to preclude extensions, or the redefining of this set to an appropriate set of attributes in particular contexts.

In general terms, fixing such domains has the effect of making more explicit knowledge available within the HLBS, thereby expanding the range of informed actions that can be taken. The decision on whether to enrich the semantics of notes resembles in its pros and cons the decision on whether to import grammatical knowledge of UIS and DBS commands into the HLBS.

The intended meanings of each of the attributes chosen are: description lets users and designers provide an abstract or summary; keyword lets users and designers tag content and thereby clarify the information contained along different categories; level lets users and designers attach a measure, e.g., of complexity, to the information; see-as-well and see-instead allow users and designers to define other hyperpages to be, respectively, comparable and alternatives to the hyperpage being annotated; wherefrom and whereto allow users and designers to provide look-back and look-ahead information so that navigational decisions can be more informed; and status lets users and designers record information about the state of the page (e.g., expiry date on time-bound information).

An example of an annotation, i.e., a well-formed string in the language defined by the grammar in Figures 5.4–5.5, is given in Figure 5.6. As the example in Figure 5.6 shows, notes can be placed with the finest granularity.
5.4.3 A Formal Grammar for the Specification of Personalisation Requests

The EBNF grammar in Figures 5.7–5.8 defines the formal syntax of a personalisation request. Note that this grammar extends (i.e., has shared non-terminals with) the grammars given in Figure 4.5 and Figures 5.4–5.5.
| **personalisation-request** | ::= | action-scope action-list |
| **action-scope** | ::= | select-page-if selection-condition |
| **selection-condition** | ::= | atomic-selection-condition |
| | | not selection-condition |
| | | selection-condition and selection-condition |
| | | selection-condition or selection-condition |
| **atomic-selection-condition** | ::= | true |
| | | atomic-selection-condition-on-page |
| | | atomic-selection-condition-on-page-part |
| | | atomic-selection-condition-on-note |
| **atomic-selection-condition-on-page** | ::= | page containment-expression |
| **atomic-selection-condition-on-page-part** | ::= | page-part containment-expression |
| **atomic-selection-condition-on-note** | ::= | relevant-note containment-expression |
| **relevant-note** | ::= | shift \(,\) note |
| **containment-expression** | ::= | contains REGULAR-EXPRESSION |
| **action-list** | ::= | action* |
| **action** | ::= | ann-then-do \{ annotation-update \} |
| | | hp-then-do \{ hp-update \} |
| **annotation-update** | ::= | update-operator condition note |
| | | update-operator note |
| | | note-projection |
| | | annotation |
| **update-operator** | ::= | insert |
| | | delete |
| **condition** | ::= | if-not QUERY-ON-ENVIRONMENT |
| | | if QUERY-ON-ENVIRONMENT |
| **note-projection** | ::= | projection-operator |
| | | selection-condition-on-note |
| **projection-operator** | ::= | drop-if |
| | | retain-if |

Figure 5.7: An EBNF Grammar of Personalisation Requests (I)
There are two parts to a personalisation request: the specification of its scope, the semantics of which is the selection of a subset of the hyperpage specifications in the hyper-library (possibly with their accompanying notes and possibly taking into account user-specific versions already stored); and the action to be performed over the hyperpage specifications in the scope.

The specification of scope is indicated by the token `select-page-if`. The set of hyperpage specifications which will be placed in scope contains every hyperpage specification that satisfies the associated selection condition. In the model, selection conditions are built up from one primitive that specifies a regular expression and is true if, and only if, the regular expression occurs in the specified page, page-part or note occurrence. Primitives can be combined using Boolean connectives to form complex selection conditions. There is also a vacuously true selection condition that caters for the need to select an entire set.
The possible actions on hyperpage annotations are:

- updates on notes (insert or delete), with which annotations can be built and maintained one by one;
- projections on sets of notes (drop-if or retain-if), with which one can filter a set of notes using a selection condition.

The persistence of notes can be made conditional on the environment, in that they may be stipulated to hold only if a condition on the environment holds (or does not hold). Furthermore, as a Note is always associated with at least one hyperpage and the identification of a set of hyperpages is determined by the selection condition, the EBNF grammar need not make a distinction between the parameters associated with the update operators insert and delete.

Possible actions on hyperpage specifications are:

- actions on hyperpage structures, which resemble those on notes in that they can be updates or projections based on selection conditions;
- actions on terminal strings, (i.e., post-composition rewrites).

All actions on hyperpage specifications are destructive and their effect is manifested the next time the H-region composition function is invoked to parse and interpret them.

Note that the only annotation that has an effect on the renderable text produced by a personalised hyperpage specification is a post-composition rewrite. All other notes are either informative and/or simply determine navigational choices.
5.4.4 Examples of Personalisation Requests

A few examples of personalisation requests, i.e., well-formed strings in the language defined by the grammar in Figures 5.7–5.8, are now given with comments on their effect.

Example 1

```
select-page-if
  true
hp-then-do {
  insert [1, chunk (R-spec)]
    "These pages belong to Goldsmiths College"
}
ann-then-do {
  insert page: "GB" -> "UK";
}
```

The personalisation request above applies to all hyperpages since the condition for selection is vacuously true of any hyperpage. The effect on each selected hyperpage is as follows: the string “These pages belong to Goldsmiths College” is inserted into the R-spec of the first chunk, then each occurrence within the hyperpage of the string “GB” is replaced with the string “UK”.

Example 2

```
select-page-if
  page contains "1997"
hp-then-do {
  delete [2, chunk]
}
```

The personalisation request above applies to all hyperpages that contain the string “1997”. The effect on each selected hyperpage is the deletion of its second chunk.
Example 3

select-page-if
    not [5, chunk] contains 'secret'
hp-then-do {
    delete [5, chunk]
    insert [5, chunk]
        chunk {
            content { X := 'This page is not classified.' }
            rendering { [<I>] X [<*/I>]}
        }
}

The personalisation request above applies to all hyperpages that do not contain the string “secret”. The effect on each selected hyperpage is the replacement of its fifth chunk by the one given above as argument to the insert operation.

Example 4

select-page-if
    [any, chunk (any, entry-point)] contains 'www.cs.gold.ac.uk'
ann-then-do {
    insert [any, chunk (any, entry-point)]: 'cs' -> 'cis'
}

The personalisation request above applies to all hyperpages in which there exists an entry-point to a chunk, such that this entry point contains the string “www.cs.gold.ac.uk”. The effect on each selected hyperpage is the replacement of each occurrence in entry-points of the string “cs” with the string “cis”.

Example 5

select-page-if
    true
hp-then-do {
    retain-if [any, chunk] contains '1998'
}

The personalisation request above applies to all hyperpages. The effect on each selected hyperpage is the deletion of any chunk in which the string “1998” does not occur.
Example 6

```plaintext
select-page-if
  true
ann-then-do {
  insert [1, chunk]: level := 1;
}
```

The personalisation request above applies to all hyperpages. The effect on each selected hyperpage is the insertion of a note in the annotation corresponding to that hyperpage defining its first chunk to have level 1.

This section has defined the formal elements of the P-region. In section 5.5, the formal semantics of personalisation requests is presented.

### 5.5 Formal Semantics of Personalisation Requests

This section formally defines the semantics of personalisation requests. The approach taken is now briefly described.

A parser has been developed\(^2\) to describe the analysis of the structure and meaning of the language for personalisation requests defined in subsection 5.4.3. The parser consists of a lexical analyser, a syntactic analyser and a semantic analyser. The lexical analyser groups the individual characters of a personalisation request specification into tokens. The syntactic analyser parses the phrase structure generated by the lexical analyser to determine whether the stream of tokens forms a valid personalisation request. If a personalisation request is validated, then the lexical analyser outputs its abstract syntax tree. Such a tree illustrates the validation process performed. Semantic analysis is the assignment of meaning. This is achieved by mapping the abstract syntax tree of a personalisation request into a target language, whose meaning it is assumed the audience (or machine) know.

Broadly, the meaning of personalisation requests can be understood as requests to manipulate hyperpage specifications and hyperpage annotations. To express the meaning of a personalisation request, it is advantageous to model hyperpage specifications using a data structure that is well understood and for which there exists a well understood language for manipulating data modelled using such structures. The relational model of data [Mannila and Räähä, 1994] is a formalism based on a single, simple data structure called the `relation`.

\(^2\)See chapter 7 for a detailed discussion on the development of the parser for hyperpage specifications, hyperpage annotations and personalisation requests.
Relational algebra is a formalism for writing queries to databases that conform to the relational data model. Relational algebraic operations can be categorised into retrievals, which do not change the database state, and updates, which do. For the purpose of defining the meaning of a personalisation request, the hyper-library has been modelled using the relational data model. Following this, a recursive function is defined that traverses the abstract syntax tree of a personalisation request, generating the corresponding semantic representation as a sequence of relational algebraic expressions and assignments.

Appendix F defines the basic concepts which underpin the relational data model and are used to represent hyperpages and hyperpage annotations. This appendix also briefly describes the relational algebra used to operate on relations.

The remainder of this section is structured as follows. Subsection 5.5.1 details the representation of hyperpages and hyperpage annotations in the relational model. In subsection 5.5.3, the semantic function is defined and possible extensions described. Subsection 5.5.4 then presents several examples of personalisation programs generated by the semantic function.

5.5.1 Hyperpage and Hyperpage Annotation Relational Schemas

This subsection defines first the domains and then the relational schemas used to model hyperpages and hyperpage annotations. Hyperpages and hyperpage annotations are input into a hyper-library as text files. It is assumed that the implementation of the hyper-library which stores hyperpages and hyperpage annotations provides the functionality of a modern database system.

The transformation of hyperpage and hyperpage annotations text files into relation schemas is achieved by retrieving in sequence each hyperpage (or hyperpage annotation) and then analysing its structure. This analysis involves reading each hyperpage from top to bottom and then transforming each part of its structure (i.e., entry-points, C-specs, R-specs and exit-points) into appropriate relations. When reconstructed as relations each part of the structure of a hyperpage (or hyperpage annotation) is given an extra attribute value as a unique identifier. This identifier acts as the primary key for that relation. Identifiers are created in the same sequence as the transformation of hyperpages (or hyperpage annotations). As a side effect these identifiers also hold the ordering in which page parts were transformed and stored into their relations. Note that the definitions of the operations to project, select and join relations given in appendix F are restricted so that they may not affect the ordering of the tuples in the relations used as their parameters. The process outlined above is assumed to be implementation specific and therefore is not formally defined further.
Domains for Hyperpages and Hyperpage Annotations

This subsection defines the domains used to model hyperpages and hyperpage annotations. Recall that domain names are denoted by a sequence of upper-case SANS-SERIF letters.

**Definition (Values).** A value is an atomic element (e.g., integers, characters). Let \( V \) denote the set of all values.

Values are assumed to be representable as in modern programming languages.

**Definition (Template Variables).** A template variable is a variable name. Let \( T \) denote the set of all template variables.

Template variables correspond, syntactically, to identifiers in modern computer languages.

**Definition (Regular Expression).** Let \( \sum \) denote an alphabet, then a regular expression, \( r \), over \( \sum \), defines a language as follows:

1. \( \emptyset \) is a regular expression, corresponding to the empty language \( \emptyset \).
2. \( \Lambda \) is a regular expression corresponding to the language consisting of the empty sequence.
3. For each symbol \( a \in \sum \), \( a \) is a regular expression corresponding to the language \( \{a\} \).
4. For any regular expressions \( r \) and \( s \) over \( \sum \), corresponding to the languages \( L_r \) and \( L_s \), respectively, each of the following is a regular expression over \( \sum \), corresponding to the language indicated.
   - \( (r, s) \), corresponding to \( L_r L_s \) where \( L_r L_s = \{xy | x \in L_r \land y \in L_s\} \)
   - \( (r + s) \), corresponding to \( L_r \cup L_s \)
   - \( (r^*) \), corresponding to \( L_r^* \) where \( * \) denotes closure.
5. Only expressions that can be produced using parts 1 - 4 are regular expressions over \( \sum \). Let \( \text{RE} \) denote the set of all regular expressions.

**Definition (UIS Strings).** A user interface server string (UIS string) is an expression in a language which a UIS can render. Let \( U \) denote the set of all UIS strings.

**Definition (DB Strings).** A database server string (DBS String) is an expression in a language which a DBS can evaluate. Let \( \text{DS} \) denote the set of all DBS strings.

**Definition (Content Assignments).** A content assignment is a pair \( <t, e> \) where \( t \in T \) and \( e \in V \cup \text{DS} \). Let \( A \) denote the set of all content assignments.

**Definition (Rendering Elements).** Let \( r \in T \cup U \) denote a rendering element. Let \( R \) denote the set of all rendering elements.
Definition (Entry Points). An entry point is a sequence \( u_1, \ldots, u_n \) where \( n \geq 1 \) and each \( u_i \in U \). Let \( P\downarrow \) denote the set of all entry points.

Definition (Exit Points). An exit point is a sequence \( u_1, \ldots, u_n \) where \( n \geq 1 \) and each \( u_i \in U \). Let \( P\uparrow \) denote the set of all exit points.

Definition (C-specs). A C-spec is a sequence \( a_1, \ldots, a_n \) where \( n \geq 0 \) and each \( a_i \in A \). Let \( CS \) denote the set of all C-specs.

Definition (R-specs). A R-spec is a sequence \( t_1, \ldots, t_n \) where \( n \geq 0 \) and each \( t_i \in R \). Let \( RS \) denote the set of all R-specs.

Definition (Chunks). A chunk is a quadruple \( \langle \vec{p}\downarrow, cs, rs, \vec{p}\uparrow \rangle \) where \( cs \in CS, rs \in RS, \vec{p}\downarrow \) is a sequence \( p_1 \downarrow, \ldots, p_n \downarrow \), with \( n \geq 0 \) and each \( p_i \downarrow \in P\downarrow \), and \( \vec{p}\uparrow \) is a sequence \( p_1 \uparrow, \ldots, p_m \uparrow \) with \( m \geq 0 \) and each \( p_i \uparrow \in P\uparrow \). Let \( C \) denote the set of all chunks.

Definition (Pages). A page is a sequence \( c_1, \ldots, c_n \) where \( n \geq 0 \) and each \( c_i \in C \). Let \( P \) denote the set of all pages.

Definition (Note Attributes). A note attribute, \( at \), is an identifier for an attribute. Let \( AT \) denote the set of all note attributes.

Definition (Attribute Assignments). An attribute assignment is a pair \( <a, v> \) where \( a \in AT \) and \( v \in V \). Let \( AA \) denote the set of all attribute assignments.

Definition (Rewrite Assignments). A rewrite assignment is a pair \( <re, re'> \) where \( re, re' \in RE \). Let \( RA \) denote the set of all attribute assignments.

Definition (Notes). A note is an element of \( AA \cup RA \). Let \( N \) denote the set of all notes.

Definition (Annotations). An annotation is a sequence \( n_1, \ldots, n_m \) where \( m \geq 0 \) and each \( n_i \in N \). Let \( ANN \) denote the set of all annotations.

Relation Schemas for Hyperpages and Hyperpage Annotations

Hyper-library relation schemas are now defined. Relation names are denoted by a sequence of upper-case SANS-SERIF letters, the corresponding name in lower-case sans-serif letters denotes an identifier of an entity modelled by the relation. The primary key of each relation is underlined. Some remarks are provided to increase readability.

\[
\text{PAGE}(\text{page, chunk, shift})
\]

The relation PAGE has the following attributes: page uniquely identifies a page, chunk identifies a chunk within that page, shift represents the position of that chunk within the page. chunk is a foreign key to CHUNK.
CHUNK\((chunk,\text{\texttt{entpointset}},c\text{-spec},r\text{-spec},\text{\texttt{exitpointset}})\)

In the relation CHUNK the entrypointset attribute identifies the set of entry points for that chunk, c-spec identifies the C-spec for the chunk, r-spec identifies the R-spec for the chunk, exitpointset identifies the set of exit points for the chunk.

ENTRYPOINTSET\((\text{\texttt{entpointset}},\text{\texttt{shift}},\text{\texttt{entpoint}})\)

In the relation ENTRYPOINTSET the entpoint attribute uniquely identifies an entry point for that entry point set and is a foreign key to ENTRYPOINT.

ENTRYPOINT\((\text{\texttt{entpoint}},\text{\texttt{e\_string}})\)

In the relation ENTRYPOINT the e_string attribute stores an entry point.

C-SPEC\((c\text{-spec}, \text{\texttt{shift}},c\text{-element})\)

In the relation C-SPEC the c-element attribute uniquely identifies an element within that C-spec and is a foreign key to C-ELEMENT.

C-ELEMENT\((c\text{-element},\text{\texttt{variable}},c\text{-string})\)

In the relation C-ELEMENT the variable attribute stores a template variable and the c_string stores a content expression. Recall that a content expression may be either a DBS string or a value.

R-SPEC\((r\text{-spec}, \text{\texttt{shift}},r\text{-element})\)

In the relation R-SPEC the r-element attribute uniquely identifies an element within that R-spec and is a foreign key to R-ELEMENT.

R-ELEMENT\((r\text{-element},r\text{-string})\)

In the relation R-ELEMENT the r_string attribute stores a rendering element. Recall that a rendering element may be either a template variable or a UIS string.

EXITPOINTSET\((\text{\texttt{exitpointset}},\text{\texttt{shift}},\text{\texttt{exitpoint}})\)

In the relation EXITPOINTSET the exitpoint attribute uniquely identifies an exit point for that exit point set and is a foreign key to EXITPOINT.

EXITPOINT\((\text{\texttt{exitpoint}},x\text{-string})\)

In the relation EXITPOINT the x_string attribute stores an exit point.

Personalisation requests are based on annotating and rewriting the hyperpage specifica-
tions. An annotation pairs a hyperpage specification with notes of interest to the user. Notes take one of the following forms. Firstly, a note can assign user-specific values to user-generic attributes of interest. Secondly, a note can specify a rewriting action over the renderable text after it has been composed i.e, after content has been fetched and made ready for display.

\[
\text{NOTE}(\text{note}, \text{page}, \text{chunk}, \text{n\_type}, \text{scope}, \text{shift}, \text{lhs}, \text{n\_string}, \text{condition})
\]

The relation \text{NOTE} has the following attributes: \text{note} uniquely identifies a note, \text{page} identifies the page that note has been associated with and is a foreign key to the relation \text{PAGE}, \text{chunk} identifies a chunk within that page and is a foreign key to the \text{CHUNK}, \text{n\_type} represents the type of note (i.e., attribute-assignment, or rewrite), \text{scope} represents the scope of a note which may be a page or a page part and \text{shift} represents the position of that page part, \text{lhs} stores a note attribute or, in the case of a rewrite note, the regular expression which is the subject of the rewrite, \text{n\_string} stores the note itself or, in the case of a rewrite note the replacement string, \text{condition} stores a query on the state of the environment.

\[
\text{ANNOTATION}(\text{annotation}, \text{page})
\]

The relation \text{ANNOTATION} has the following attributes: \text{annotation} uniquely identifies an annotation, \text{page} identifies the page that an annotation has been associated with and is a foreign key to the relation \text{PAGE}.

### 5.5.2 Preliminary Definitions

Parsing is the process used to describe the analysis of the structure of expressions of a language. Parsing consists of lexical analysis, syntactic analysis and semantic analysis. Lexical analysis is the grouping of individual characters of a source language program into tokens. Syntactic analysis is the parsing of the phrase structure of the source language program to determine whether the stream of tokens from the lexical analyser forms a valid sentence in the source language grammar. The output is a syntax tree. Semantic analysis is the assignment of meaning. This is achieved by mapping the abstract syntax into a formalism (target language) of which the target audience (or machine) knows the meaning.

In this and the following subsection a recursive function is defined that traverses an abstract syntax tree, represented as a term\(^3\), generating the corresponding semantic representation of a personalisation request.

Informally a personalisation request may be partitioned into a specification of its \textit{scope}, whose semantics is the selection of a subset of the hyperpage specifications in the

\(^3\text{In section 7.3 a detailed overview is given of the parsing process for personalisation requests including the generation of term representations from abstract syntax trees}\)
hyper-library and the action to be performed over the hyperpage specifications which fall within the scope. To define the semantics of personalisation requests the following preliminary definitions are needed.

**Definition (Ordering Tuples).** Let $\omega_{A_1,\ldots,A_n}(R)$ denote the sequence of tuples derived by ordering $R$ alphabetically by $A_1 > \ldots > A_n$.

Therefore $R' = \pi_A(\omega_{A_1,\ldots,A_n}(R))$ is a ordered list of tuples derived by first ordering $R$ alphabetically then projecting on the attribute $A$, where $A \in R$.

**Definition (String Concatenation).** Assume that the attribute $A$ of a unary relation $R$ has values drawn from the domain of strings, and let $a_i$ denote the value of $A$ in the $i$-th tuple of $R$, $0 \leq i \leq |R|$. Then let $\hat{\downarrow}(R) = a_1 \ldots a_{|R|}$ be the string obtained by concatenating in sequence all the strings in $R$ that appear as a value for $A$. The concatenation of the string-valued attributes $A_1,\ldots,A_n$ into a single string is denoted by $A_1\ldots A_n$.

**Definition (String Reconstruction).** Given a $n$-ary relation $R(A_1,\ldots,A_n)$, let $\langle a_1,\ldots,a_n \rangle$ be a subset $R$ if, resp., $a_1,\ldots,a_n$ are the values in $A_1,\ldots,A_n$ for some tuple $t \in R$. If for some tuple $t$ the value of one or more of the attributes is not relevant in the context then this is denoted by a wild-card character ‘∗’. The string-reconstruction $[h]$ of a hyperpage (or hyperpage part) $h$ is defined recursively as follows:

1. If $h$ is the entrypoint identified by $e$ then

   $$[e] = \hat{\downarrow}((\pi_{e \text{string}}(\sigma_{entrypoint=e} \text{ENTRYPOINT}))$$

   The above expression first selects a subset of entry points, then a projection is performed on the resulting relation yielding a new relation consisting of a single column $e_{\text{string}}$. Finally the values in this column are concatenated. The remaining cases have a similar definition.

2. If $h$ is the entrypointset identified by $E$ then

   $$[E] = \hat{\downarrow}((\pi_{e \text{string}}(\text{ENTRYPOINT } *_{entrypoint} \pi_{entrypoint}(\omega_{shift}(\sigma_{entrypointset=E} \text{ENTRYPOINTSET}))))))$$

3. If $h$ is the C-element identified by $c$ then

   $$[c] = \hat{\downarrow}((\pi_{variable \text{c_string}}(\sigma_{c-element=c} \text{C-ELEMENT}))$$

4. If $h$ is the C-spec identified by $C$ then

   $$[C] = \hat{\downarrow}((\pi_{variable \text{c_string}}(\text{C-ELEMENT } *_{c-element}(\pi_{c-element}(\omega_{shift}(\sigma_{c-spec=C} \text{C-SPEC}))))))$$
5. If $h$ is the R-element identified by $r$ then

$$[r] = \downarrow (\pi_{r\text{-string}}(\sigma_{r\text{-element}=r}\text{R-ELEMENT}))$$

6. If $h$ is the R-spec identified by $R$ then

$$[R] = \downarrow (\pi_{r\text{-string}}(\text{R-ELEMENT} \ast_{r\text{-element}} (\pi_{r\text{-element}}(\omega_{\text{shift}}(\sigma_{r\text{-spec}=R}\text{R-SPEC}))))))$$

7. If $h$ is the exitpoint identified by $x$ then

$$[x] = \downarrow (\pi_{x\text{-string}}(\sigma_{\text{exitpoint}=x}\text{EXITPOINT}))$$

8. If $h$ is the exitpointset identified by $X$ then

$$[X] = \downarrow (\pi_{x\text{-string}}(\text{EXITPOINT} \ast_{\text{exitpoint}} (\pi_{\text{exitpoint}}(\omega_{\text{shift}}(\sigma_{\text{exitpointset}=X}\text{EXITPOINTSET}))))))$$

9. If $h$ is the chunk identified by $K$ then, for some $E, C, R, X$, $(K, E, C, R, X) \in \text{CHUNK}$, and

$$[K] = [\text{entrypointset}(K)] \cap [\text{c-spec}(K)] \cap [\text{r-spec}(K)] \cap [\text{exitpointset}(K)]$$

where $\text{entrypointset}(K)=E$, $\text{c-spec}(K)=C$, $\text{r-spec}(K)=R$ and $\text{exitpointset}(K)=X$.

10. If $h$ is the page identified by $P$ then

$$[P] = \downarrow (\pi_{\text{chunk}}(\omega_{\text{shift}}(\sigma_{\text{page}=P}\text{PAGE})))$$

where, if $R$ is the sequence of tuples $\langle k_1, \ldots, k_{|R|} \rangle$ then $\downarrow (R) = [k_1] \ldots [k_{|R|}]$

11. If $h$ is the note identified by $n$ then

$$[n] = \downarrow (\pi_{n\text{-string}}(\sigma_{\text{note}=n}\text{NOTE}))$$

To improve readability, the following meta-linguistic abbreviations are defined:

1. Given two strings $s$ and $s'$, $s \prec s'$ if $s$ occurs in (e.g., is a substring of) $s'$.

2. Given a regular expression $E$, let $L(E)$ denote the set of strings defined by $E$. An expression of the form ‘$\exists s \in L(E). s \prec |h|$’ (where $h$ is the identifier of a hyperpage or a hyperpage part) is abbreviated by ‘$E \prec |h|$’.  

5.5.3 Semantics

Given the successful parsing of a personalisation request whose output is a term representation of the syntax tree, $T$, of the personalisation request, the semantics of $T$, denoted by
\[T\] is recursively defined in section 5.5.3. The meaning of an action list, AL is defined as a function \([AL]\) which takes a parameter \(P\) of type \(page\). \([AL]\)(\(P\)) is written to denote the application of \([AL]\) to the parameter \(P\).

**Notational Conventions and Assumptions**

In the definitions that follow, a shift, in the grammar for hyperpages and hyperpage annotations, is assumed to be represented as a signed integer. These definitions can be extended to include the case where a shift is the token \(all\), but to aid clarity this is not discussed here. Section 5.5.3 outlines how such extensions can be made. Furthermore, a shift value denoted by \(shift\), may be sub- or superscripted.

“\(E_1 ; E_2\)” is written to mean that \(E_1\) is performed and then \(E_2\) is performed.

“\(\text{foreach } P \in S \text{ do } E \text{ endfor}\)” is written to mean that \(E\) is performed for each \(P\) in the finite set \(S\).

“\(\text{NOP}\)” is written to mean no operation is performed.

“\(\text{if } X \text{ then } E \text{ else NOP endif}\)”, where \(X\) is a Boolean value, is written to mean that if \(X\) is \(true\) then \(E\) is performed else no operation is performed.

“\(\text{for } i := 1 \text{ to } n \text{ do } E \text{ endfor}\)” is written to mean that \(E\) is performed with \(i = 1\), then with \(i = 2\) and so on, up to \(i = n\).

“\(*\)” denotes all attributes in a relation.

“\(X := Y\)” is written to denote the binding of a value (or the value of an expression) \(Y\) to a denotation, \(X\).

To generate new primary keys for tuples which are to be inserted into relations, a function \(\text{New}\) is assumed that takes as a parameter the name of a relation and returns a new primary key for that relation.

To generate the next value in the order defined over an attribute domain a function \(\text{Next}\) is assumed that takes as parameters an attribute name and a relation name and returns the next value for that attribute domain. The legitimate use of the function \(\text{Next}\) is therefore restricted to ordered domains (i.e., numbers).
To generate a tuple representation of a note from the term representation of that note in an abstract syntax tree, a function Map_tuple is assumed. The function Map_tuple is formally defined in this section.

To insert a set of tuple representations of a hyperpage, hyperpage annotation or hyperpage part into a relation, a function Insert is assumed. The Insert function allows the union of a relation $R$ with a set $t$ of tuples. The insertion of a set of tuples $t$ into $R$ yields a new relation instance $R'$. ‘$R' = \text{Insert}(R,t)$’ is written to denote the insertion of tuple $t$ into relation $R$.

To delete a set of tuple representations of a hyperpage, hyperpage annotation or hyperpage part from a relation a function Delete is assumed. The Delete function allows the difference between a relation $R$ and a set $t$ of tuples to be obtained. The deletion of the tuple $t$ from the relation $R$ will yield a new relation instance $R'$. ‘$R' = \text{Delete}(R,t)$’ is written to denote the deletion of tuple $t$ from relation $R$.

**Meaning: Personalisation Request**

Given the relation PAGE, a personalisation_request selects into a unary relation the primary keys of the tuples specified in the action-scope and applies a set of personalisation actions to each page referenced by each primary key in the unary relation one at a time.

\[
\text{if } T = \text{‘personalisation_request(AS,AL)’} \\
\text{then} \quad [[T]] = \text{‘foreach } P \in [[AS]] \text{ do } [[AL]](P) \text{ endfor’}]
\]

**Meaning: Action Scope**

Given the token select_page_if and a selection condition, action_scope selects into a unary relation the primary keys of those tuples in the PAGE relation that satisfy the specified selection condition.

\[
\text{if } T = \text{‘action_scope(select-page-if,C)’} \\
\text{then} \quad [[T]] = [[C]]
\]

\(^4\)See appendix F for an overview of the relational data model and relational algebra.
Meaning: Selection Conditions

Given an atomic_selection_condition and a set of tuples, each tuple (i.e., page) is tested to see whether or not the atomic_selection_condition is satisfied. A non-atomic selection_condition may be constructed using any of the tokens not, and, or, and these are defined in the standard way.

\[
\begin{align*}
\text{if } T &= \text{`selection_condition(atomic_selection_condition(C))`} \\
&\text{then } \llbracket T \rrbracket = \llbracket C \rrbracket \\
\text{if } T &= \text{`selection_condition(not,selection_condition(C))`} \\
&\text{then } \llbracket T \rrbracket = \llbracket \pi_{\text{page}}(\sigma_{\ast \text{PAGE}} \setminus \llbracket C \rrbracket) \rrbracket \\
\text{if } T &= \text{`selection_condition(and,selection_condition(C_1),selection_condition(C_2))`} \\
&\text{then } \llbracket T \rrbracket = \llbracket C_1 \rrbracket \cap \llbracket C_2 \rrbracket \\
\text{if } T &= \text{`selection_condition(or,selection_condition(C_1),selection_condition(C_2))`} \\
&\text{then } \llbracket T \rrbracket = \llbracket C_1 \rrbracket \cup \llbracket C_2 \rrbracket
\end{align*}
\]

Meaning: Atomic Selection Condition true

Given the token true, an atomic_selection_condition selects all the primary keys of all the tuples in the relation PAGE.

\[
\begin{align*}
\text{if } T &= \text{`atomic_selection_condition(true)`} \\
&\text{then } \llbracket T \rrbracket = \text{`}\pi_{\text{page}}(\sigma_{\ast \text{PAGE}})\text{`}
\end{align*}
\]

Meaning: An Atomic Selection Condition on a Page

Given an atomic_selection_condition_on_page whose parameter is the token page, and a containment_expression, whose parameters are the token contains and a regular expression, EXP, the primary keys of tuples that contain the specified regular expression are selected.

\[
\begin{align*}
\text{if } T &= \text{`atomic_selection_condition(atomic_selection_condition_on_page}
\text{ (page, containment_expression[contains,EXP]))`} \\
&\text{then } \llbracket T \rrbracket = \text{`}\pi_{\text{page}}(\sigma_{\text{EXP} \triangledown \text{pagePAGE}})\text{`}
\end{align*}
\]
Meaning: Atomic Selection Condition on a Page Part

Given an atomic_selection_condition_on_page_part whose parameters are a sequence of values and tokens denoting a specified page-part i.e. shift, chunk, entry-point, c-spec, r-spec, exit-point, and a containment_expression whose parameters are the token contains and a regular expression EXP, the primary keys of tuples that contain the specified regular expression are selected.

if $T = \text{atomic_selection_condition}(\text{atomic_selection_condition_on_page_part}$

$\quad\text{page_part}((\text{relevant_chunk}(\text{shift}, \text{chunk})),\text{containment_expression}(\text{contains},\text{EXP}))')$

then

$[[T]] = ' \pi_{\text{page}}(\sigma_{\text{shift}=\text{shift}}(\text{PAGE} \Join_{\text{chunk}} (\pi_{\text{chunk}}(\sigma_{\text{EXP}<\text{chunk}} \text{CHUNK})))')$

if $T = \text{atomic_selection_condition}(\text{atomic_selection_condition_on_page_part}$

$\quad\text{page_part}((\text{relevant_chunk}(\text{shift}_1, \text{chunk}, (\text{relevant_multi_part}(\text{shift}_2, \text{entry-point}))),$

$\quad\text{containment_expression}(\text{contains},\text{EXP}))')$

then

$[[T]] = ' \pi_{\text{page}}(\sigma_{\text{shift}=\text{shift}_1}(\text{PAGE} \Join_{\text{chunk}} (\pi_{\text{chunk}}(\text{CHUNK} \Join_{\text{entrypointset}}$

$\quad(\pi_{\text{entrypointset}}(\sigma_{\text{shift}=\text{shift}_2 \land \text{EXP}<\text{entrypoint}} \text{ENTRYPOINTSET})))))')$

if $T = \text{atomic_selection_condition}(\text{atomic_selection_condition_on_page_part}$

$\quad\text{page_part}((\text{relevant_chunk}(\text{shift}, \text{chunk}, (\text{relevant_single_part}(\text{c-spec}))),$

$\quad\text{containment_expression}(\text{contains},\text{EXP}))')$

then

$[[T]] = ' \pi_{\text{page}}(\sigma_{\text{shift}=\text{shift}_1}(\text{PAGE} \Join_{\text{chunk}} (\pi_{\text{chunk}}(\text{CHUNK} \Join_{\text{c-spec}}$

$\quad(\pi_{\text{c-spec}}(\sigma_{\text{EXP}<\text{c-spec}} \text{C-SPEC}))))))')$

if $T = \text{atomic_selection_condition}(\text{atomic_selection_condition_on_page_part}$

$\quad\text{page_part}((\text{relevant_chunk}(\text{shift}, \text{chunk}, (\text{relevant_single_part}(\text{r-spec}))),$

$\quad\text{containment_expression}(\text{contains},\text{EXP}))')$

then

$[[T]] = ' \pi_{\text{page}}(\sigma_{\text{shift}=\text{shift}_1}(\text{PAGE} \Join_{\text{chunk}} (\pi_{\text{chunk}}(\text{CHUNK} \Join_{\text{r-spec}}$

$\quad(\pi_{\text{r-spec}}(\sigma_{\text{EXP}<\text{r-spec}} \text{R-SPEC}))))))')$

if $T = \text{atomic_selection_condition}(\text{atomic_selection_condition_on_page_part}$

$\quad\text{page_part}((\text{relevant_chunk}(\text{shift}_1, \text{chunk}, (\text{relevant_multi_part}(\text{shift}_2, \text{exit-point}))),$

$\quad\text{containment_expression}(\text{contains},\text{EXP}))')$

then

$[[T]] = ' \pi_{\text{page}}(\sigma_{\text{shift}=\text{shift}_1}(\text{PAGE} \Join_{\text{chunk}} (\pi_{\text{chunk}}(\text{CHUNK} \Join_{\text{exitpointset}}$

$\quad(\pi_{\text{exitpointset}}(\sigma_{\text{shift}=\text{shift}_2 \land \text{EXP}<\text{exitpoint}} \text{EXITPOINTSET}))))))'$
Meaning: Atomic Selection Condition on Note

Given an atomic_selection_condition_on_page_part whose parameter is a value, \textit{shift}, which denotes a shift, the token \textit{note} and a containment expression whose parameters are the token \textit{contains} and a regular expression \textit{EXP}, the primary keys of the tuples in the relation \textit{PAGE} with notes that satisfy the specified regular expression are selected.

\[
\text{if } T = \text{atomic_selection_condition(atomic_selection_condition_on_page_part} \\
\text{(page_part(relevant_note(shift,note))), containment_expression(contains,EXP)))' then} \\
\text{[[T]]} = \pi_{\text{page}}(\text{PAGE} \star_{\text{page}} (\pi_{\text{page}}(\sigma_{\text{shift}\leq \text{shift}} \land EXP \land \text{note_string\_NOTE})))'
\]

Meaning: Action List

Given a list of actions to be performed on a hyperpage (or its associated hyperpage annotation), \textit{action\_list} applies this list of actions to the hyperpage (or its associated hyperpage annotation). \textit{action\_list} takes as a parameter the primary key of a hyperpage, \textit{P}, which is used to identify a hyperpage (or its associated hyperpage annotation). Therefore, the first action is applied to \textit{P} and then, successively, every other action in the list is applied to \textit{P}.

\[
\text{if } T = \text{action\_list(A,AL)}' \\
\text{then} \\
\text{[[T]]}_{(P)} = [[[A]]_{(P)} ; [[[AL]]}(P)]
\]

Meaning: Action

Given the token \textit{an\_then\_do} and a list of actions to be performed on hyperpage annotations, \textit{action} applies this list of actions to the annotations of the set of selected pages (the action scope) denoted by \textit{P}. Given the token \textit{hp\_then\_do} and a list of actions to be performed on hyperpages, \textit{action} applies this list of actions to the set of selected pages \textit{P}.

\[
\text{if } T = \text{action(ann\_then\_do,ATD)}' \\
\text{then} \\
\text{[[T]]}_{(P)} = [[[ATD]](P)}
\]

\[
\text{if } T = \text{action(hp\_then\_do,HTD)}' \\
\text{then} \\
\text{[[T]]}_{(P)} = [[[HTD]](P)}
\]
Given $P$ as above, an update operator denoted by the token insert (or the token delete), a condition whose parameters are the token if-not or if and a value, a query on the environment, $QOE$, and a note, $n$. The function Insert (or Delete), if the query on the environment is true in the case of the token if, (or false in the case of the token if-not) inserts (or deletes) from $P$.

if $T$ = ‘annotation_update(update_operator(insert), condition(if-not,QOE),note(n))’

then

$$[T](P) = 'if QOE = false then
\text{Insert}(\text{NOTE, Map\_tuple(note}(n),P))
\text{else}
\text{NOP}
\text{endif}'$$

if $T$ = ‘annotation_update(update_operator(insert), condition(if,QOE),note(n))’

then

$$[T](P) = 'if QOE = true then
\text{Insert}(\text{NOTE, Map\_tuple(note}(n),P))
\text{else}
\text{NOP}
\text{endif}'$$

if $T$ = ‘annotation_update(update_operator(delete), condition(if-not,QOE),note(n))’

then

$$[T](P) = 'if QOE = false then
\text{Delete}(\text{NOTE, Map\_tuple(note}(n),P,'\text{del}')
\text{else}
\text{NOP}
\text{endif}'$$

if $T$ = ‘annotation_update(update_operator(delete), condition(if,QOE),note(n))’

then

$$[T](P) = 'if QOE = true then
\text{Delete}(\text{NOTE, Map\_tuple(note}(n),P,'\text{del}')
\text{else}
\text{NOP}
\text{endif}'$$
Meaning: Annotation Update Note

Given an update_operator whose parameter is the token insert (or the token delete), and a note n, the function Insert (or Delete) inserts (or deletes) from P.

\[
\text{if } T = \text{'annotation_update(update_operator(insert),note(n))'} \\
\quad \text{then} \\
\quad \quad \llbracket T \rrbracket(P) = \text{'Insert(NOTE, \text{Map_tuple(note(n),P)})'}
\]

\[
\text{if } T = \text{'annotation_update(update_operator(delete),note(n))'} \\
\quad \text{then} \\
\quad \quad \llbracket T \rrbracket(P) = \text{'Delete(NOTE, \text{Map_tuple(note(n),P,del)})'}
\]

Meaning: Annotation Update Note Projection

Given a note_projection whose parameters are projection_operator, whose parameter is the token drop-if (or the token retain-if), and a selection_condition_on_note, annotation_update deletes a note from the relation NOTE if that note is associated with a page whose primary key is P in the relation PAGE and that in the case of drop-if meets the selection_condition_on_note and in the case of retain-if does not meet that selection_condition_on_note.

\[
\text{if } T = \text{'annotation_update(note_projection(projection_operator(drop-if)),} \\
\quad \text{selection_condition_on_note(CN))'} \\
\quad \text{then} \\
\quad \quad \llbracket T \rrbracket(P) = \llbracket \text{if } \llbracket CN \rrbracket(P) = \text{true then} \\
\quad \quad \quad \text{SN} := \pi_{\text{note}}(\text{NOTE} \bowtie \text{page}(\pi_{\text{page}}(\sigma_{\text{page}=P} \text{PAGE}))); \\
\quad \quad \quad \text{Delete} \text{NOTE, SN) else} \\
\quad \quad \quad \text{NOP} \\
\quad \text{endif} \rrbracket \\
\]

\[
\text{if } T = \text{'annotation_update(note_projection(projection_operator,retain-if)),} \\
\quad \text{selection_condition_on_note(CN))'} \\
\quad \text{then} \\
\quad \quad \llbracket T \rrbracket(P) = \llbracket \text{if } \llbracket CN \rrbracket(P) = \text{false then} \\
\quad \quad \quad \text{SN} := \pi_{\text{note}}(\text{NOTE} \bowtie \text{page}(\pi_{\text{page}}(\sigma_{\text{page}=P} \text{PAGE}))); \\
\quad \quad \quad \text{Delete} \text{NOTE, SN) else} \\
\quad \quad \quad \text{NOP} \\
\quad \text{endif} \rrbracket
\]
Meaning: Annotation Update Insert Annotation

Given the token \texttt{annotation} \texttt{annotation} \texttt{update} \texttt{insert} \texttt{annotation} inserts an annotation for a page that has the primary key $P$. A previously inserted annotation is first deleted. If $T = \texttt{annotation} \texttt{update}(\texttt{annotation})$

then

$$\llbracket T \rrbracket(P) = \text{if} \; \texttt{page} = \text{pANNOTATION} = \text{true then}$$

$$\text{Delete(ANNOTATION, } P)$$

else

$$\text{NOP}$$

endif;

$$\text{Insert(ANNOTATION, } \langle \text{New(ANNOTATION), } P \rangle)$$

Meaning: Selection Condition on Note

Given an atomic \texttt{selection} \texttt{condition} \texttt{on} \texttt{note} whose parameter is a value, \texttt{shift}, which denotes a shift, the token \texttt{note} and a containment expression whose parameters are the token \texttt{contains}, and a regular expression, $EXP$, \texttt{selection} \texttt{condition} \texttt{on} \texttt{note} returns the value \texttt{true} or \texttt{false} if a page with the primary key $P$ is associated with at least one note that satisfies the specified regular expression.

If $T = \texttt{selection} \texttt{condition} \texttt{on} \texttt{note}(\texttt{atomic} \texttt{selection} \texttt{condition} \texttt{on} \texttt{note}(\texttt{CN}))$

then

$$\llbracket T \rrbracket(P) = \llbracket \text{CN} \rrbracket(P)$$

If $T = \texttt{selection} \texttt{condition} \texttt{on} \texttt{note}(\texttt{not}\texttt{,selection} \texttt{condition} \texttt{on} \texttt{note}(\texttt{CN}))$

then

$$\llbracket T \rrbracket(P) = \neg \llbracket \text{CN} \rrbracket(P)$$

If $T = \texttt{selection} \texttt{condition} \texttt{on} \texttt{note}(\texttt{and}\texttt{,selection} \texttt{condition} \texttt{on} \texttt{note}(\texttt{CN}_1),\texttt{selection} \texttt{condition} \texttt{on} \texttt{note}(\texttt{CN}_2))$

then

$$\llbracket T \rrbracket(P) = \llbracket \text{CN}_1 \rrbracket(P) \land \llbracket \text{CN}_2 \rrbracket(P)$$

If $T = \texttt{selection} \texttt{condition} \texttt{on} \texttt{note}(\texttt{or}\texttt{,selection} \texttt{condition} \texttt{no} \texttt{note}(\texttt{CN}_1),\texttt{selection} \texttt{condition} \texttt{no} \texttt{note}(\texttt{CN}_2))$

then

$$\llbracket T \rrbracket(P) = \llbracket \text{CN}_1 \rrbracket(P) \lor \llbracket \text{CN}_2 \rrbracket(P)$$

If $T = \texttt{selection} \texttt{condition} \texttt{on} \texttt{note}(\texttt{atomic} \texttt{selection} \texttt{condition} \texttt{on} \texttt{note}$

$\texttt{(page} \texttt{part(relevant} \texttt{note(shift,note)), containment} \texttt{expression}(\texttt{contains,} EXP))))$

then

$$\llbracket T \rrbracket(P) = \text{'}\texttt{shift=}shift \land EXP<\texttt{string}\texttt{NOTE} (\texttt{NOTE} \ast \texttt{page} (\pi_{\texttt{page}}(\sigma_{\texttt{page}}=p\texttt{PAGE})))\text{'}}}$$
Meaning: Hyperpage Structure Update Insert Page Part

Given a hp\_structure\_update whose parameters are the token insert, a page\_part whose parameters are a sequence of values and tokens denoting a specified page-part and a hp\_constr whose parameter is one or more parts of a chunk as defined in the grammar for hyperpages, hp\_update inserts those parts of a chunk into a page that has as its primary key P.

if \emph{T} = 'hp\_update(hp\_structure\_update(insert, page\_part(relevant\_chunk(shift,chunk, (relevant\_multi\_part(shift,entry-point))))), hp-constr(entry-point(entry(uis))))'
then

\[ [[\text{T}]](P) = \text{entrypointset} := (\pi_{\text{entrypointset}}(\text{CHUNK} \star \text{chunk} (\pi_{\text{chunk}}(\sigma_{\text{page}=P \land shift=shift\_PAGE))));
\]
\[
\text{newentry} := \text{New}(\text{ENTRYPOINT});
\]
\[
\text{Insert}(\text{ENTRYPOINT}, \langle \text{newentry, uis} \rangle);
\]
\[
\text{Insert}(\text{ENTRYPOINTSET}, \langle \text{entrypointset, shift, newentry} \rangle)
\]

if \emph{T} = 'hp\_update(hp\_structure\_update(insert, page\_part(relevant\_chunk(shift,chunk,(relevant\_single\_part(C-spec)))))), hp-constr(C-spec(content(content\_assignment(t_1,e_1), ..., content\_assignment(t_n,e_n))))'
then

\[ [[\text{T}]](P) = \text{c-spec} := (\pi_{\text{c-spec}}(\text{C-SPEC} \star \text{chunk} (\pi_{\text{chunk}}(\sigma_{\text{page}=P \land shift=shift\_PAGE})));
\]
\[
\text{for } i := 1 \text{ to } n \text{ do}
\]
\[
\text{newelement} := \text{New}(\text{C-ELEMENT});
\]
\[
\text{Insert}(\text{C-ELEMENT}, \langle \text{newelement, t_i, e_i} \rangle);
\]
\[
\text{Insert}(\text{C-SPEC}, \langle \text{c-spec, Next(C-SPEC, shift), newelement} \rangle)
\]
\endfor

if \emph{T} = 'hp\_update(hp\_structure\_update(insert, page\_part(relevant\_chunk(shift,chunk,(relevant\_single\_part(R-spec)))))), hp-constr(R-spec(rendering(rendering\_element(r_1), ..., rendering\_element(r_n))))'
then

\[ [[\text{T}]](P) = \text{r-spec} := (\pi_{\text{r-spec}}(\text{R-SPEC} \star \text{chunk} (\pi_{\text{chunk}}(\sigma_{\text{page}=P \land shift=shift\_PAGE})));
\]
\[
\text{for } i := 1 \text{ to } n \text{ do}
\]
\[
\text{newelement} := \text{New}(\text{R-ELEMENT});
\]
\[
\text{Insert}(\text{R-ELEMENT}, \langle \text{newelement, r_i} \rangle);
\]
\[
\text{Insert}(\text{R-SPEC}, \langle \text{r-spec, Next(R-SPEC, shift), newelement} \rangle)
\]
\endfor
if $T = 'hp\_update(hp\_structure\_update(insert,
(page\_part(relevant\_chunk(shift_1,chunk,
(relevant\_multi\_part(shift_2,exit\_point))))), hp\_constr(exit\_point(exit(uis))))')$
then

$$[[T]](P) = 'exitpointset := (\pi\_{exitpointset}(CHUNK * chunk
(\pi\_chunk(\sigma_{page=P \land shift=shift\_PAGE}))));
newexit := New(EXITPOINT);
Insert(EXITPOINT, \langle newexit, uis \rangle);
Insert(EXITPOINTSET, \langle exitpointset, shift_2, newexit \rangle)'$$
Meaning: Hyperpage Structure Update Insert Chunk

Given a `hp_structure_update` whose parameters are the token `insert`, a `page_part` whose parameters are a sequence of tokens denoting a specified chunk and a `hp_constr` whose parameter is a chunk as defined in the grammar for hyperpages, `hp_update` inserts that chunk into a page that has the primary key `P`.

if \( T = \text{hp_update}(\text{hp_structure_update}(\text{insert}, \text{page_part}(\text{relevant_chunk}(\text{shift}, \text{chunk})), \text{hp-constr}(\text{entry-point}(\text{entry}(uis_1), \ldots, \text{entry}(uis_n)), (\text{C-spec}(\text{content}(t_1, e_1), \ldots, \text{content}(t_m, e_m)))), (\text{R-spec}(\text{rendering}(\text{rendering_element}(r_1), \ldots, \text{rendering_element}(r_m)))))), (\text{exit-point}(\text{exit}(uis_1), \ldots, \text{exit}(uis_v)))))'  
then

\[
\begin{align*}
[T](P) &= [\pi_{\text{entrypointset}}(\text{CHUNK} \star_{\text{chunk}} \\
(\pi_{\text{chunk}}(\sigma_{\text{page}=P \land \text{shift}=\text{shift_PAGE}))))];
\text{for } i := 1 \text{ to } n \text{ do } \\
\text{  } & \quad \text{newentry} := \text{New}(\text{ENTRYPOINT}); \\
\text{  } & \quad \text{Insert}(\text{ENTRYPOINT}, (\text{newentry}, uis_i)); \\
\text{  } & \quad \text{Insert}(\text{ENTRYPOINTSET}, (\text{entrypointset}, \text{Next}(\text{ENTRYPOINTSET}, \text{shift}), \text{newentry}))
\end{align*}
\]

\[
\begin{align*}
\text{c-spec} &:= (\text{C-SPEC} \star_{\text{chunk}} (\pi_{\text{chunk}}(\sigma_{\text{page}=P \land \text{shift}=\text{shift_PAGE}))))];
\text{for } i := 1 \text{ to } m \text{ do } \\
\text{  } & \quad \text{newcelement} := \text{New}(\text{C-ELEMENT}); \\
\text{  } & \quad \text{Insert}(\text{C-ELEMENT}, (\text{newcelement}, t_i, e_i)); \\
\text{  } & \quad \text{Insert}(\text{C-SPEC}, (\text{c-spec}, \text{Next}(\text{C-SPEC}, \text{shift}), \text{newcelement}))
\end{align*}
\]

\[
\begin{align*}
\text{r-spec} &:= (\text{R-SPEC} \star_{\text{chunk}} (\pi_{\text{chunk}}(\sigma_{\text{page}=P \land \text{shift}=\text{shift_PAGE}))))];
\text{for } i := 1 \text{ to } m \text{ do } \\
\text{  } & \quad \text{newrelement} := \text{New}(\text{R-ELEMENT}); \\
\text{  } & \quad \text{Insert}(\text{R-ELEMENT}, (\text{newrelement}, t_i, e_i)); \\
\text{  } & \quad \text{Insert}(\text{R-SPEC}, (\text{r-spec}, \text{Next}(\text{R-SPEC}, \text{shift}), \text{newrelement}))
\end{align*}
\]

\[
\begin{align*}
\text{exitpointset} &:= (\pi_{\text{exitpointset}}(\text{CHUNK} \star_{\text{chunk}} \\
(\pi_{\text{chunk}}(\sigma_{\text{page}=P \land \text{shift}=\text{shift_PAGE}))))];
\text{for } i := 1 \text{ to } v \text{ do } \\
\text{  } & \quad \text{newexit} := \text{New}(\text{EXITPOINT}); \\
\text{  } & \quad \text{Insert}(\text{EXITPOINT}, (\text{newexit}, uis_i)); \\
\text{  } & \quad \text{Insert}(\text{EXITPOINTSET}, (\text{exitpointset}, \text{Next}(\text{EXITPOINTSET}, \text{shift}), \text{newexit}))
\end{align*}
\]
Meaning: Hyperpage Structure Update Delete Page Part

Given a hp\_structure\_update whose parameters are the token delete and a page\_part whose parameters are a sequence of tokens denoting a specified page-part hp\_update deletes that page-part from a page that has the primary key \( P \).

\[
\text{if } T = 'hp\_update(hp\_structure\_update(delete, (page\_part (relevant\_chunk(shift, chunk)))))' \text{ then }
\]
\[
\text{[[T]]}(P) = 'SEP:=(CHUNK \star_{chunk} (\pi_{chunk}(\sigma_{page=\overline{P}} \wedge shift=\overline{shift \_PAGE}))); Delete(CHUNK, SC)'
\]

\[
\text{if } T = 'hp\_update(hp\_structure\_update(delete, (page\_part (relevant\_chunk(shift_1,chunk,(relevant\_multi\_part(shift_2,entry-point)))))))' \text{ then }
\]
\[
\text{[[T]]}(P) = 'SP:=(\sigma_{shift=\overline{shift_2}}(ENTRYPOINT \star_{entrypoint} (\pi_{entrypoint}(ENTRYPOINTSET \star_{entrypointset} (\pi_{entrypointset}(CHUNK \star_{chunk} (\pi_{chunk}(\sigma_{page=\overline{P}} \wedge shift=\overline{shift_1 \_PAGE}))))))); Delete(ENTRYPOINT,SEP)'
\]

\[
\text{if } T = 'hp\_update(hp\_structure\_update(delete, (page\_part (relevant\_chunk(shift,chunk,(relevant\_single\_part(C\_spec)))))'))' \text{ then }
\]
\[
\text{[[T]]}(P) = 'SCS:=(C\_SPEC \star_{C\_spec} (\pi_{C\_spec}(CHUNK \star_{chunk} (\pi_{chunk}(\sigma_{page=\overline{P}} \wedge shift=\overline{shift_1 \_PAGE})))))); Delete(C\_SPEC, SCS)'
\]

\[
\text{if } T = 'hp\_update(hp\_structure\_update(delete, (page\_part (relevant\_chunk(shift,chunk,(relevant\_single\_part(R\_spec)))))'))' \text{ then }
\]
\[
\text{[[T]]}(P) = 'SRS:=(R\_SPEC \star_{R\_spec} (\pi_{R\_spec}(CHUNK \star_{chunk} (\pi_{chunk}(\sigma_{page=\overline{P}} \wedge shift=\overline{shift_1 \_PAGE})))))); Delete(R\_SPEC, SRS)'
\]

\[
\text{if } T = 'hp\_update(hp\_structure\_update(delete, page\_part (relevant\_chunk(shift_1,chunk,(relevant\_multi\_part(shift_2,exit-point)))))))' \text{ then }
\]
\[
\text{[[T]]}(P) = 'SEXP:=(\sigma_{shift=\overline{shift_2}}(EXITPOINT \star_{exitpoint} (\pi_{exitpoint}(EXITPOINTSET \star_{exitpointset} (\pi_{exitpointset}(CHUNK \star_{chunk} (\pi_{chunk}(\sigma_{page=\overline{P}} \wedge shift=\overline{shift_1 \_PAGE}))))))); Delete(EXITPOINT,SEXP)'
\]
Meaning: Hyperpage Structure Update Projection

Given a hp structure update whose parameters are a projection operator whose parameter is the token drop-if (or the token retain-if), and a selection condition on hp constr, hp update deletes a page that has the primary key P from the relation PAGE if that page in the case of drop-if meets the selection condition on hp constr and in the case of retain-if does not meet that selection condition on hp constr.

if T = ‘hp update(hp structure update(projection operator(drop-if)), selection condition on hp constr(CHC))’
  then
      [[T]](P) = [[if [CHC](P) = true then
      SP := (σpage=PAGE);
      Delete(PAGE, SP)
      else
      NOP
      endif’]]

if T = ‘hp update(hp structure update(projection operator(retain-if)), selection condition on hp constr(CHC))’
  then
      [[T]](P) = [[if [CHC](P) = false then
      SP := (σpage=PAGE);
      Delete(PAGE, SP)
      else
      NOP
      endif’]]
Meaning: Selection Condition on Hyperpage Construct

Given an atomic \texttt{selection\_condition\_on\_hp\_constr}, \texttt{selection\_condition\_on\_hp\_constr} tests a page with the primary key \( P \) to see whether or not the atomic \texttt{selection\_condition\_on\_hp\_constr} is satisfied. A \texttt{selection\_condition\_on\_hp\_constr} may be preceded by any of the tokens \texttt{not}, \texttt{and}, \texttt{or}, and these are defined in the standard way.

\begin{align*}
\text{if } T &= \text{‘selection\_condition\_on\_hp\_constr(atomic\_selection\_condition\_on\_hp\_constr(CHC))’} \\
&\quad \text{then} \\
[T](P) &= [CHC](P) \\
\text{if } T &= \text{‘selection\_condition\_on\_hp\_constr(not,selection\_condition\_on\_hp\_constr(CHC))’} \\
&\quad \text{then} \\
[T](P) &= \neg [CHC](P) \\
\text{if } T &= \text{‘selection\_condition\_on\_hp\_constr(and,} \\
&\quad \text{selection\_condition\_on\_hp\_constr(CHC\_1),selection\_condition\_on\_hp\_constr(CHC\_2))’} \\
&\quad \text{then} \\
[T](P) &= [CHC\_1](P) \land [CHC\_2](P) \\
\text{if } T &= \text{‘selection\_condition\_on\_hp\_constr(or,selection\_condition\_on\_hp\_constr(CHC\_1),} \\
&\quad \text{selection\_condition\_no\_hp\_constr(CHC\_2))’} \\
&\quad \text{then} \\
[T](P) &= [CHC\_1](P) \lor [CHC\_2](P)
\end{align*}
Meaning: Atomic Selection Condition on Hyperpage Construct

Given a sequence of values and tokens denoting a specified page-part i.e. shift, chunk, entry-point, c-spec, r-spec, exit-point, and a containment expression whose parameters are the token contains, and a regular expression, EXP, atomic_selection_condition_on_hp_constr returns true (or false) if a page with the primary key P has a specified page-part that contains the specified regular expression.

if T = ‘atomic_condition_on_hp_constr(page_part(relevant_chunk(shift, chunk))), containment_expression(contains, EXP))’

then

\[
\left\lbrack T \right\rbrack(P) = \left\lbrack EXP \land chunk(CHUNK \star chunk(\pi_{chunk}(\sigma_{\text{shift} \land \text{page} = P\text{PAGE})))) \right\rbrack
\]

if T = ‘atomic_condition_on_hp_constr(page_part(relevant_chunk(shift1, chunk), (relevant_multi_part(shift2, entry-point))), containment_expression(contains, EXP))’

then

\[
\left\lbrack T \right\rbrack(P) = \left\lbrack \text{string}\text{ENTRYPOINTSET}(\text{ENTRYPOINTSET} \star \text{entrypointset}
(\pi_{\text{entrypointset}}(\text{CHUNK} \star chunk(\pi_{chunk}(\sigma_{\text{shift} \land \text{page} = P\text{PAGE}})))) \right\rbrack
\]

if T = ‘atomic_condition_on_hp_constr(page_part(relevant_chunk(shift, chunk), (relevant_single_part(c-spec))), containment_expression(contains, EXP))’

then

\[
\left\lbrack T \right\rbrack(P) = \left\lbrack EXP \land c\text{-spec}\text{C-SPEC}(C\text{-SPEC} \star c\text{-spec}
(\pi_{\text{c-spec}}(\text{CHUNK} \star chunk(\pi_{chunk}(\sigma_{\text{shift} \land \text{page} = P\text{PAGE}})))) \right\rbrack
\]

if T = ‘atomic_condition_on_hp_constr(page_part(relevant_chunk(shift, chunk), (relevant_single_part(r-spec))), containment_expression(contains, EXP))’

then

\[
\left\lbrack T \right\rbrack(P) = \left\lbrack EXP \land r\text{-spec}\text{R-SPEC}(R\text{-SPEC} \star r\text{-spec}
(\pi_{\text{r-spec}}(\text{CHUNK} \star chunk(\pi_{chunk}(\sigma_{\text{shift} \land \text{page} = P\text{PAGE}})))) \right\rbrack
\]

if T = ‘atomic_condition_on_hp_constr(page_part(relevant_chunk(shift1, chunk), (relevant_multi_part(shift2, exit-point))), containment_expression(contains, EXP))’

then

\[
\left\lbrack T \right\rbrack(P) = \left\lbrack \text{string}\text{EXITPOINTSET}(EXITPOINTSET \star \text{exitpointset}
(\pi_{\text{exitpointset}}(\text{CHUNK} \star chunk(\pi_{chunk}(\sigma_{\text{shift} \land \text{page} = P\text{PAGE}})))) \right\rbrack
\]
Meaning: Hyperpage Terminal Rewrite

Given a `hp_terminal_rewrite` whose parameters are a `rewrite` whose parameter is a note, `n`. `hp_update` inserts a rewrite note into the relation `NOTE` for a page with the primary key `P`. The note inserted specifies a rewriting action over the renderable text after it has been composed by the H-region, i.e., after content has been fetched and made ready for display (e.g., to map American into British spelling forms).

If a hyperpage is associated with a note that specifies a rewriting action over that hyperpage, after it has been composed, this rewriting action is applied to that page. The functionality required such rewriting is exhibited by commonly available programs such as the unix stream editor `sed`.\footnote{sed is a stream editor used to perform basic text transformations on an input stream (a file or input from a pipeline). sed works by making one pass over the input(s) in order to filter text.} [Ray, 1999].
Map to Tuple Function

The following section details the function \textbf{Map\_tuple} which is called in the main body of the Meaning function.

Given an abstract syntax tree which denotes a specified page-part and a note, and a primary key of a page, \( P \), the function \textbf{Map\_tuple} returns a tuple of type note for the specified page-part.

\[
\text{Map\_tuple}(\text{note}(\text{scope}(\text{page})),\text{attribute\_assignment}(\text{attribute}(a),\text{attribute\_value}(v)))', P) = \text{‘newnote:=New(NOTE)’}
\]
\[
\quad\text{Return (newnote, } P, \text{null, attrib, page, null, } a, v, \text{null)}
\]

\[
\text{Map\_tuple}(\text{note}(\text{scope}(\text{page\_part}(\text{relevant\_chunk}(\text{shift},\text{chunk}))),\text{attribute\_assignment}(\text{attribute}(a),\text{attribute\_value}(v))), P) = \text{‘newnote:=New(NOTE)’}
\]
\[
\quad\text{chunk:= } (\pi_{\text{chunk}}(\sigma_{\text{shift=shift\&\ page=PAGE}}))'
\]
\[
\quad\text{Return (newnote, } P, \text{chunk, attrib, chunk, null, } a, v, \text{null)}
\]

\[
\text{Map\_tuple}(\text{note}(\text{scope}(\text{page\_part}(\text{relevant\_chunk}(\text{shift},\text{chunk})),\text{relevant\_multi\_part}(\text{shift2, entry\_point}))),\text{attribute\_assignment}(\text{attribute}(a),\text{attribute\_value}(v))), P) = \text{‘newnote:=New(NOTE)’}
\]
\[
\quad\text{chunk:= } (\pi_{\text{chunk}}(\sigma_{\text{shift=shift\&\ page=PAGE}}))'
\]
\[
\quad\text{Return (newnote, } P, \text{chunk, attrib, entry, shift2, } a, v, \text{null)}
\]

\[
\text{Map\_tuple}(\text{note}(\text{scope}(\text{page\_part}(\text{relevant\_chunk}(\text{shift},\text{chunk})),\text{relevant\_single\_part}(\text{C-spec}))),\text{attribute\_assignment}(\text{attribute}(a),\text{attribute\_value}(v))), P) = \text{‘newnote:=New(NOTE)’}
\]
\[
\quad\text{chunk:= } (\pi_{\text{chunk}}(\sigma_{\text{shift=shift\&\ page=PAGE}}))'
\]
\[
\quad\text{Return (newnote, } P, \text{chunk, attrib, cspec, null, } a, v, \text{null)}
\]

\[
\text{Map\_tuple}(\text{note}(\text{scope}(\text{page\_part}(\text{relevant\_chunk}(\text{shift},\text{chunk})),\text{relevant\_single\_part}(\text{R-spec}))),\text{attribute\_assignment}(\text{attribute}(a),\text{attribute\_value}(v))), P) = \text{‘newnote:=New(NOTE)’}
\]
\[
\quad\text{chunk:= } (\pi_{\text{chunk}}(\sigma_{\text{shift=shift\&\ page=PAGE}}))'
\]
\[
\quad\text{Return (newnote, } P, \text{chunk, attrib, rspec, null, } a, v, \text{null)}
\]

\[
\text{Map\_tuple}(\text{note}(\text{scope}(\text{page\_part}(\text{relevant\_chunk}(\text{shift1},\text{chunk})),\text{relevant\_multi\_part}(\text{shift2, exit\_point}))),\text{attribute\_assignment}(\text{attribute}(a),\text{attribute\_value}(v))), P) = \text{‘newnote:=New(NOTE)’}
\]
\[
\quad\text{chunk:= } (\pi_{\text{chunk}}(\sigma_{\text{shift=shift\&\ page=PAGE}}))'
\]
\[
\quad\text{Return (newnote, } P, \text{chunk, attrib, exit, shift2, } a, v, \text{null)}
\]

Figure 5.9: Function: \textbf{Map\_tuple} (I)
Map_tuple('note(scope(page),attribute_assignment(attribute(a),attribute_value(v)))', P;'del')
= Return ⟨{*}, P, null, attrib, page, null, a, v, null⟩

Map_tuple('note(scope(page_part(relevant_chunk(shift,chunk))), attribute_assignment(attribute(a),attribute_value(v)))', P;'del')
= 'chunk:= (π_{chunk}(σ_{shift=shift\&\page=P\\text{PAGE}}))'
Return ⟨{*}, P, chunk, attrib, chunk, null, a, v, null⟩

Map_tuple('note(scope(page_part(relevant_chunk(shift,chunk)), relevant_multi_part(shift_2,entry-point))), attribute_assignment(attribute(a),attribute_value(v)))', P;'del')
= 'chunk:= (π_{chunk}(σ_{shift=shift\&\page=P\\text{PAGE}}))'
Return ⟨{*}, P, chunk, attrib, entry, shift_2, a, v, null⟩

Map_tuple('note(scope(page_part(relevant_chunk(shift,chunk)),relevant_single_part(C-spec))), attribute_assignment(attribute(a),attribute_value(v)))', P;'del')
= 'chunk:= (π_{chunk}(σ_{shift=shift\&\page=P\\text{PAGE}}))'
Return ⟨{*}, P, chunk, attrib, cspec, null, a, v, null⟩

Map_tuple('note(scope(page_part(relevant_chunk(shift,chunk)),relevant_single_part(R-spec))), attribute_assignment(attribute(a),attribute_value(v)))', P;'del')
= 'chunk:= (π_{chunk}(σ_{shift=shift\&\page=P\\text{PAGE}}))'
Return ⟨{*}, P, chunk, attrib, rspec, null, a, v, null⟩

Map_tuple('note(scope(page_part(relevant_chunk(shift_1,chunk)), relevant_multi_part(shift_2,exit-point))), attribute_assignment(attribute(a),attribute_value(v)))', P;'del')
= 'chunk:= (π_{chunk}(σ_{shift=shift\&\page=P\\text{PAGE}}))'
Return ⟨{*}, P, chunk, attrib, exit, shift_2, a, v, null⟩

Figure 5.10: Function: Map_tuple (II)
\textbf{Map\_tuple}\\
\begin{quote}
\textbf{Map\_tuple}
(\text{\texttt{\textbf{note(rewrite(scope(page,from(EXP1),to(EXP2)))}},P}}) \\
\quad = \text{Return } \langle \text{newnote, P, null, rewrite, page, null, EXP1, EXP2, null} \rangle
\end{quote}

\textbf{Map\_tuple}\\
\begin{quote}
\textbf{Map\_tuple}
(\text{\texttt{\textbf{note(rewrite(scope(page\_part(relevant\_chunk(shift,chunk))),from(EXP1),to(EXP2)))}},P}) \\
\quad = \langle \text{\texttt{\textbf{newnote:=New(NOTE)}}} \rangle \\
\quad \quad \quad \texttt{chunk:= (π\_chunk(σ\_shift=shift\_Page=pPage))} \\
\quad \quad \quad \text{Return } \langle \text{\texttt{\textbf{newnote, P, chunk, rewrite, chunk, null, EXP1, EXP2, null}}} \rangle
\end{quote}

\textbf{Map\_tuple}\\
\begin{quote}
\textbf{Map\_tuple}
(\text{\texttt{\textbf{note(rewrite(scope(page\_part(relevant\_chunk(shift,chunk),relevant\_multi\_part(shift2,entry\_point))),from(EXP1),to(EXP2)))}},P}) \\
\quad = \langle \text{\texttt{\textbf{newnote:=New(NOTE)}}} \rangle \\
\quad \quad \quad \texttt{chunk:= (π\_chunk(σ\_shift=shift\_Page=pPage))} \\
\quad \quad \quad \text{Return } \langle \text{\texttt{\textbf{newnote, P, chunk, rewrite, entry, shift2, EXP1, EXP2, null}}} \rangle
\end{quote}

\textbf{Map\_tuple}\\
\begin{quote}
\textbf{Map\_tuple}
(\text{\texttt{\textbf{note(rewrite(scope(page\_part(relevant\_chunk(shift,chunk),relevant\_single\_part(C\_spec))),from(EXP1),to(EXP2)))}},P}) \\
\quad = \langle \text{\texttt{\textbf{newnote:=New(NOTE)}}} \rangle \\
\quad \quad \quad \texttt{chunk:= (π\_chunk(σ\_shift=shift\_Page=pPage))} \\
\quad \quad \quad \text{Return } \langle \text{\texttt{\textbf{newnote, P, chunk, rewrite, cspec, null, EXP1, EXP2, null}}} \rangle
\end{quote}

\textbf{Map\_tuple}\\
\begin{quote}
\textbf{Map\_tuple}
(\text{\texttt{\textbf{note(rewrite(scope(page\_part(relevant\_chunk(shift,chunk),relevant\_single\_part(R\_spec))),from(EXP1),to(EXP2)))}},P}) \\
\quad = \langle \text{\texttt{\textbf{newnote:=New(NOTE)}}} \rangle \\
\quad \quad \quad \texttt{chunk:= (π\_chunk(σ\_shift=shift\_Page=pPage))} \\
\quad \quad \quad \text{Return } \langle \text{\texttt{\textbf{newnote, P, chunk, rewrite, rspec, null, EXP1, EXP2, null}}} \rangle
\end{quote}

\textbf{Map\_tuple}\\
\begin{quote}
\textbf{Map\_tuple}
(\text{\texttt{\textbf{note(rewrite(scope(page\_part(relevant\_chunk(shift1,chunk),relevant\_multi\_part(shift2,exit\_point))),from(EXP1),to(EXP2)))}},P}) \\
\quad = \langle \text{\texttt{\textbf{newnote:=New(NOTE)}}} \rangle \\
\quad \quad \quad \texttt{chunk:= (π\_chunk(σ\_shift=shift\_Page=pPage))} \\
\quad \quad \quad \text{Return } \langle \text{\texttt{\textbf{newnote, P, chunk, rewrite, exit, shift2, EXP1, EXP2, null}}} \rangle
\end{quote}

Figure 5.11: Function: Map\_tuple(III)
\begin{align*}
\text{Map\_tuple}(\text{note}(\text{rewrite}(\text{scope}(\text{page}), \text{from}(\text{EXP}1), \text{to}(\text{EXP}2))), \text{P}, \text{del})
& \quad = \text{Return} (\ast, \text{P}, \text{null}, \text{rewrite}, \text{page}, \text{null}, \text{EXP}1, \text{EXP}2, \text{null}) \\
\text{Map\_tuple}(\text{note}(\text{rewrite}(\text{scope}(\text{page}\_\text{part}(\text{relevant}\_\text{chunk}(\text{shift}, \text{chunk}))), \text{from}(\text{EXP}1), \text{to}(\text{EXP}1))), \text{P}, \text{del})
& \quad = \text{chunk}:= (\pi_{\text{chunk}}(\sigma_{\text{shift}\land \text{page}=\text{P}\text{PAGE}}))' \\
& \quad \quad \text{Return} (\ast, \text{P}, \text{chunk}, \text{rewrite}, \text{chunk}, \text{null}, \text{EXP}1, \text{EXP}2, \text{null}) \\
\text{Map\_tuple}(\text{note}(\text{rewrite}(\text{scope}(\text{page}\_\text{part}(\text{relevant}\_\text{chunk}(\text{shift}, \text{chunk}), \text{relevant}\_\text{multi}\_\text{part}(\text{shift2}, \text{entry-point}))), \text{from}(\text{EXP}1), \text{to}(\text{EXP}2))), \text{P}, \text{del})
& \quad = \text{chunk}:= (\pi_{\text{chunk}}(\sigma_{\text{shift}\land \text{page}=\text{P}\text{PAGE}}))' \\
& \quad \quad \text{Return} (\ast, \text{P}, \text{chunk}, \text{rewrite}, \text{entry}, \text{shift2}, \text{EXP}1, \text{EXP}2, \text{null}) \\
\text{Map\_tuple}(\text{note}(\text{rewrite}(\text{scope}(\text{page}\_\text{part}(\text{relevant}\_\text{chunk}(\text{shift}, \text{chunk}), \text{relevant}\_\text{single}\_\text{part}(\text{C-spec}))), \text{from}(\text{EXP}1), \text{to}(\text{EXP}2))), \text{P}, \text{del})
& \quad = \text{chunk}:= (\pi_{\text{chunk}}(\sigma_{\text{shift}\land \text{page}=\text{P}\text{PAGE}}))' \\
& \quad \quad \text{Return} (\ast, \text{P}, \text{chunk}, \text{rewrite}, \text{cspec}, \text{null}, \text{EXP}1, \text{EXP}2, \text{null}) \\
\text{Map\_tuple}(\text{note}(\text{rewrite}(\text{scope}(\text{page}\_\text{part}(\text{relevant}\_\text{chunk}(\text{shift}, \text{chunk}), \text{relevant}\_\text{single}\_\text{part}(\text{R-spec}))), \text{from}(\text{EXP}1), \text{to}(\text{EXP}2))), \text{P}, \text{del})
& \quad = \text{chunk}:= (\pi_{\text{chunk}}(\sigma_{\text{shift}\land \text{page}=\text{P}\text{PAGE}}))' \\
& \quad \quad \text{Return} (\ast, \text{P}, \text{chunk}, \text{rewrite}, \text{rspec}, \text{null}, \text{EXP}1, \text{EXP}2, \text{null}) \\
\text{Map\_tuple}(\text{note}(\text{rewrite}(\text{scope}(\text{page}\_\text{part}(\text{relevant}\_\text{chunk}(\text{shift}, \text{chunk}), \text{relevant}\_\text{multi}\_\text{part}(\text{shift2}, \text{exit-point}))), \text{from}(\text{EXP}1), \text{to}(\text{EXP}2))), \text{P}, \text{del})
& \quad = \text{chunk}:= (\pi_{\text{chunk}}(\sigma_{\text{shift}\land \text{page}=\text{P}\text{PAGE}}))' \\
& \quad \quad \text{Return} (\ast, \text{P}, \text{chunk}, \text{rewrite}, \text{exit}, \text{shift2}, \text{EXP}1, \text{EXP}2, \text{null})
\end{align*}

Figure 5.12: Function: \texttt{Map\_tuple(IV)}
\textbf{Map\_tuple}('note(QOE, rewrite(scope(page), from(EXP_1), to(EXP_2))), P)
= Return (\text{newnote}, P, \text{null}, rewrite, page, \text{null}, EXP_1, EXP_2, QOE)

\textbf{Map\_tuple}('note(QOE, rewrite(scope(page_part(relevant\_chunk(shift, chunk))), from(EXP_1), to(EXP_2))), P)
= '\text{newnote}:=\text{New}(\text{NOTE})
\text{chunk}:= (\pi_{\text{chunk}}(\sigma_{\text{shift} \land \text{page}=\text{pPAGE}}))'
Return (\text{newnote}, P, \text{chunk}, rewrite, \text{chunk}, \text{null}, EXP_1, EXP_2, QOE)

\textbf{Map\_tuple}('note(QOE, rewrite(scope(page_part(relevant\_chunk(shift, chunk), relevant\_multi\_part(shift_2, entry\_point))), from(EXP_1), to(EXP_2))), P)
= '\text{newnote}:=\text{New}(\text{NOTE})
\text{chunk}:= (\pi_{\text{chunk}}(\sigma_{\text{shift} \land \text{page}=\text{pPAGE}}))'
Return (\text{newnote}, P, \text{chunk}, rewrite, \text{chunk}, \text{null}, EXP_1, EXP_2, QOE)

\textbf{Map\_tuple}('note(QOE, rewrite(scope(page_part(relevant\_chunk(shift, chunk), relevant\_single\_part(C\_spec))), from(EXP_1), to(EXP_2))), P)
= '\text{newnote}:=\text{New}(\text{NOTE})
\text{chunk}:= (\pi_{\text{chunk}}(\sigma_{\text{shift} \land \text{page}=\text{pPAGE}}))'
Return (\text{newnote}, P, \text{chunk}, rewrite, \text{cspec}, \text{null}, EXP_1, EXP_2, QOE)

\textbf{Map\_tuple}('note(QOE, rewrite(scope(page_part(relevant\_chunk(shift, chunk), relevant\_single\_part(R\_spec))), from(EXP_1), to(EXP_2))), P)
= '\text{newnote}:=\text{New}(\text{NOTE})
\text{chunk}:= (\pi_{\text{chunk}}(\sigma_{\text{shift} \land \text{page}=\text{pPAGE}}))'
Return (\text{newnote}, P, \text{chunk}, rewrite, \text{rspec}, \text{null}, EXP_1, EXP_2, QOE)

\textbf{Map\_tuple}('note(QOE, rewrite(scope(page_part(relevant\_chunk(shift_1, chunk), relevant\_multi\_part(shift_2, exit\_point))), from(EXP_1), to(EXP_2))), P)
= '\text{newnote}:=\text{New}(\text{NOTE})
\text{chunk}:= (\pi_{\text{chunk}}(\sigma_{\text{shift} \land \text{page}=\text{pPAGE}}))'
Return (\text{newnote}, P, \text{chunk}, rewrite, \text{exit}, \text{shift_2}, EXP_1, EXP_2, QOE)

Figure 5.13: Function: \textbf{Map\_tuple}(V)
Map_tuple('note(QOE,rewrite(scope(page), from(EXP_1), to(EXP_2))), P, 'del')
  = Return (newnote, P, null, rewrite, page, null, EXP_1, EXP_2, QOE)

Map_tuple('note(QOE,rewrite(scope(page_part(relevant_chunk(shift, chunk))), from(EXP_1), to(EXP_1)), P, 'del')
  = 'chunk':= (π_chunk(σ_shift=shift ∧ page=P PAGE))’
    Return (newnote, P, chunk, rewrite, chunk, null, EXP_1, EXP_2, QOE)

Map_tuple('note(QOE,rewrite(scope(page_part(relevant_chunk(shift, chunk), relevant_multi_part(shift_2, entry-point))), from(EXP_1), to(EXP_2)), P, 'del')
  = 'chunk':= (π_chunk(σ_shift=shift ∧ page=P PAGE))’
    Return (newnote, P, chunk, rewrite, entry, shift_2, EXP_1, EXP_2, QOE)

Map_tuple('note(QOE,rewrite(scope(page_part(relevant_chunk(shift, chunk), relevant_single_part(C-spec))), from(EXP_1), to(EXP_2)), P, 'del')
  = 'chunk':= (π_chunk(σ_shift=shift ∧ page=P PAGE))’
    Return (newnote, P, chunk, rewrite, cspec, null, EXP_1, EXP_2, QOE)

Map_tuple('note(QOE,rewrite(scope(page_part(relevant_chunk(shift, chunk), relevant_single_part(R-spec))), from(EXP_1), to(EXP_2)), P, 'del')
  = 'chunk':= (π_chunk(σ_shift=shift ∧ page=P PAGE))’
    Return (newnote, P, chunk, rewrite, rspec, null, EXP_1, EXP_2, QOE)

Map_tuple('note(QOE,rewrite(scope(page_part(relevant_chunk(shift_1, chunk), relevant_multi_part(shift_2, exit-point))), from(EXP_1), to(EXP_2)), P, 'del')
  = 'chunk':= (π_chunk(σ_shift=shift ∧ page=P PAGE))’
    Return (newnote, P, chunk, rewrite, exit, shift_2, EXP_1, EXP_2, QOE)

Figure 5.14: Function: Map_tuple(VI)
Extending the Meaning Function

In the definitions for the Meaning function provided above a shift in the grammar for hyperpages and hyperpage annotations is assumed to represent a signed integer. The following subsection provides details of how to extend these definitions to include the case where a shift is the token any. Informally, all that is required is to remove the constraint that a tuple refers to a specific part of a page at a specific shift (position). The effect of these is to refer to all parts rather than a specific subset.

For example the case

\[
\text{if } T = \text{atomic selection condition(atomic selection condition on page part page part((relevant chunk(shift, chunk))), containment expression(contains, EXP))}
\]

then

\[
[[T]] = \pi_{\text{page}}(\sigma_{\text{shift}=\text{shift}}(\text{PAGE} \ast\text{chunk}\ (\pi_{\text{chunk}}(\sigma_{\text{EXP}<\text{chunk}}\text{CHUNK})))))
\]

would become

\[
\text{if } T = \text{atomic selection condition(atomic selection condition on page part page part((relevant chunk(shift, chunk))), containment expression(contains, EXP))}
\]

then

\[
[[T]] = \pi_{\text{page}}(\text{PAGE} \ast\text{chunk}\ (\pi_{\text{chunk}}(\sigma_{\text{EXP}<\text{chunk}}\text{CHUNK})))
\]

and the case

\[
\text{if } T = \text{atomic selection condition(atomic selection condition on page part page part(relevant chunk(shift, chunk), (relevant multi part(shift, entry-point)), containment expression(contains, EXP))}
\]

then

\[
[[T]] = \pi_{\text{page}}(\sigma_{\text{shift}=\text{shift}}(\text{PAGE} \ast\text{chunk}\ (\pi_{\text{chunk}}\text{CHUNK} \ast\text{entrypointset }\ (\pi_{\text{entrypointset}}(\sigma_{\text{shift}=\text{shift} \&\text{EXP}<\text{entrypoint}}\text{ENTRYPOINTSET}))))))
\]

would become

\[
\text{if } T = \text{atomic selection condition(atomic selection condition on page part page part(relevant chunk(shift, chunk), (relevant multi part(shift, entry-point)), containment expression(contains, EXP))}
\]

then

\[
[[T]] = \pi_{\text{page}}(\text{PAGE} \ast\text{chunk}\ (\pi_{\text{chunk}}(\text{CHUNK} \ast\text{entrypointset }\ (\pi_{\text{entrypointset}}(\sigma_{\text{EXP}<\text{entrypoint}}\text{ENTRYPOINTSET}))))))
\]
It is hoped that the examples shown here illustrate how the semantic function could be extended to deal with the case where the shift in the grammar for hyperpages and hyperpage annotations is not a signed integer but is the token \textit{any}.

### 5.5.4 Examples of Personalisation Request Programs

This section provides examples of how, given the successful parsing of the personalisation request examples shown in subsection 5.4.4, the semantics of such requests may be represented as a mapping into relational algebra with assignments.

```plaintext
select-page-if
  true
hp-then-do {
  insert [1, chunk (R-spec)]
  "These pages belong to Goldsmiths College"
}
ann-then-do {
  insert page : "GB" -> "UK";
}

'foreach $P \in \pi_{\text{page}}(\sigma_{\ast \text{PAGE}})\ '$ do
  'r-spec := (\pi_{\text{r-spec}}(\text{R-SPEC} * \text{chunk}(\pi_{\text{chunk}}(\sigma_{\text{page}=P \land \text{shift}=1}\text{PAGE})))));
for $i := 1$ to $1$ do 
  newrelement:=\text{New}(\text{R-ELEMENT});
  \text{Insert(}R\text{-ELEMENT},\langle\text{newrelement,\"These pages belong to Goldsmiths College\"}\rangle);
  \text{Insert(}R\text{-SPEC},\langle\text{r-spec,\text{Next}(R\text{-SPEC,shift),\text{newrelement})}\rangle)
endfor'
  'Insert(\text{NOTE},\langle\text{newnote, P, null, rewrite, page, null, \"UK\", \"GB\", null}\rangle)'
endfor'

Figure 5.15: Personalisation Program Example 1

```plaintext
select-page-if
  page contains "1997"
hp-then-do {
  delete [2, chunk]
}

'foreach $P \in \pi_{\text{page}}(\sigma_{1997 < \text{page}\text{PAGE}})\ '$ do
  'SC:=\text{CHUNK} * \text{chunk}(\pi_{\text{chunk}}(\sigma_{\text{page}=P \land \text{shift}=2}\text{PAGE})));
  \text{Delete(}CHUNK,SC)'
endfor'

Figure 5.16: Personalisation Program Example 2
```
select-page-if
not [5, chunk] contains 'secret'
hp-then-do {
dele [5, chunk]
insert [5, chunk]
chunk {
  content { X := 'This page is not classified.' }
  rendering { [<I>] X [</I>] }
}
}

foreach P ∈ \(\pi_{\text{page}}(\sigma_{\ast \text{PAGE}})\)
  'SC:=(\text{CHUNK} \ast \text{chunk} (\pi_{\text{chunk}}(\sigma_{\ast \text{PAGE}})))''
do
  Delete(CHUNK, SC)
  'c-spec:= (\pi_{\text{c-spec}}(\text{C-SPEC} \ast \text{chunk} (\pi_{\text{chunk}}(\sigma_{\ast \text{PAGE}}))));
  for i := 1 to 1 do
    newclement:=New(C-ELEMENT);
    Insert(C-ELEMENT,⟨newclement, X, "This page is not classified."⟩);
    Insert(C-SPEC, ⟨c-spec, Next(C-SPEC, shift), newclement⟩)
  endfor
  'r-spec:= (\pi_{\text{r-spec}}(\text{R-SPEC} \ast \text{chunk} (\pi_{\text{chunk}}(\sigma_{\ast \text{PAGE}}))));
  for i := 1 to 1 do
    newrelement:=New(R-ELEMENT);
    Insert(R-ELEMENT,⟨newrelement, \("[<I>]X[</I>]"\)⟩);
    Insert(R-SPEC,⟨r-spec, Next(R-SPEC, shift), newrelement⟩)
  endfor
endfor

Figure 5.17: Personalisation Program Example 3

select-page-if
[any, chunk (any, entry-point)] contains 'www.cs.gold.ac.uk'
ann-then-do {
  insert page: 'cs' -> 'cis'
}

foreach P ∈ \(\pi_{\text{page}}(\pi_{\text{chunk}}(\pi_{\text{chunk}}(\sigma_{\ast \text{PAGE}}) \ast \text{chunkset} (\pi_{\text{chunkset}}(\sigma_{\ast \text{ENTRYPOINTSET}}))))\) do
  'Insert(NOTE,⟨newnote, P, null, rewrite, page, null,"cs", "cis", null⟩)'
endfor

Figure 5.18: Personalisation Program Example 4

137
select-page-if
  true
hp-then-do {
  retain-if [any, chunk] contains ‘‘1998’’
}

‘foreach $P \in \pi_{\text{page}}(\sigma_{\text{PAGE}})$ do
  ‘if 
  ‘if 
  ‘else
  NOP
endif’
endfor’

Figure 5.19: Personalisation Program Example 5

select-page-if
  true
ann-then-do {
  insert [1, chunk]: level := 1;
}

‘foreach $P \in \pi_{\text{page}}(\sigma_{\text{PAGE}})$ do
  ‘Insert(\text{NOTE},\text{newnote}, P, \text{chunk}, \text{attrib, chunk}, 1, “level”, “1”,\text{null})’
endfor’

Figure 5.20: Personalisation Program Example 6
5.6 Opportunities Afforded by the P-Region

The abstract, additive model for personalising hyperlink-based interaction described in this chapter is one answer to the question of which P&A actions could be made available to users. It qualifies as one such answer insofar as it indicates how, given a conceptual framework such as that which underlies the H-region, it is possible to devise a language for personalisation whose effects fall out as a consequence of that very conceptual framework being adopted.

The language for personalisation is induced from the formal definition of the emergent properties of HLBSs. It is complete, in the sense that it is expressive enough to override all design decisions, and hence can be said to be an instrument for the transfer of ownership of every aspect of the interaction with a HLBS.

Attention is drawn to the fact that even as the detail of the model could be presented in equivalent ways, the methodological procedure of isolating the scope for P&A, defining it formally and then inducing from that formalisation the set of P&A actions needed, is a contribution that can be made use of in other settings, under different assumptions and using alternative conceptualisations of HLBSs.

In this section, it is shown how the formal characterisation of hyperpage specifications induces a set of P&A actions that allows a user to override all the design decisions that a hyperpage specification embodies. It is shown that the P-region provides the functionality needed for P&A actions without disrupting and, in fact, relying on the functionality of the H-region remaining intact and unaltered.

Specifically the model has shown how all personalisation techniques described in [Brusilovsky, 1996b] may be formally modelled and therefore understood. For example, the technique of link annotation [Brusilovsky and Pesin, 1995, Calvi and de Bra, 1997] (the representation of availability of links and the incorporation of visual clues, indicating their status or purpose) can be clearly represented in the model (see example 1).

Example 1

```
annotation{
    [7, chunk (5, exit-point) ]: visible := yes;
    [7, chunk (5, exit-point) ]: keyword := 'introduction';
}
```

The notion of not only controlling the visualisation of links but also the ability to manipulate them (i.e., allowing them to be specified according to the appropriateness of the situation [Calvi and de Bra, 1997] can also be represented (see Example 2).
Example 2

```plaintext
select-page-if
  [any, chunk (1, entry-point)] contains "www.cs.gold.ac.uk"
ann-then-do {
  [any, chunk (1, entry-point)]: "cs" -> "cis"
}
```

Example 3 below illustrates how classical personalisation techniques, such as the personalisation of content [Boyle and Encarnacion, 1994, Höök et al., 1996], can be clearly represented within the model.

Example 3

```plaintext
select-page-if
  page contains "Computer Science"
hp-then-do {
  delete [1, chunk]
}
```

Example 4 shows how selective content (i.e., hiding parts of information about a particular concept which the user has expressed a wish not to see) can be modelled.

Example 4

```plaintext
select-page-if
  [1, chunk] contains "Goldsmiths"
hp-then-do {
  delete [1, chunk]
}
```

Hyperpage annotations may also be used to model comparative and variable content as implemented in [Kay and Kummerfeld, 1995, Hohl et al., 1996] (see Example 5).

Example 5

```plaintext
annotation{
  page : see-as-well := [http://www.Gold.co.uk/]
  [1, chunk (C-spec)]: see-as-well := [http://www.bt.com/index.htm/]
}
```

Example 6 shows that within the P-region it is also possible to model the incorporation of additional explanations for particular concepts found on a hyperpage. Such
functionality is provided by systems such as KN-AHS [Kobsa et al., 1994] and Anatom-Tutor [Beaumont, 1994].

Example 6

```
annotation{
    page : level := 1;
    page : see-as-well := [http://www.bbc.com/]
    [any, chunk ]: keyword := 'dynamic content,'
    [1, chunk ]: keyword := 'introduction,'
    [2, chunk (C-spec) ]: keyword := 'UK Television';
}
```

Although not explicitly addressed in the model presented, approaches to prerequisite content selection (supplementing a request for content with additional content which describes all prerequisite concepts related to the request) and directed guidance [Brusilovsky and Pesin, 1994] could be modelled using the P-region if a topology were introduced which allowed for the notion of a hyperdocument (i.e., a designer-imposed structure on collections of hyperpages). The issue of designer-imposed structures on collections of hyperpages is an important one but it is also orthogonal to the model and therefore not addressed in this research.

In summary, the abstract model of personalisable hyperlink-based interaction contributed by this chapter possesses the following beneficial characteristics:

1. **All design decisions are subject to personalisation** All decisions designers of hyperdocuments make with regard to content, structure and presentation of units of information are able to be the subject of P&A actions;

2. **Complete ownership transfer** The conceptual framework of the P-region allows for a complete transfer of ownership of the hyperdocument from its designers to its users;

3. **Explicit P&A actions** All P&A actions are modelled explicitly and are formally defined. One should note that although a fixed set of attributes is modelled for hyperpages to illustrate their use, there is nothing to preclude the extensions of this set to appropriate attributes in particular contexts;

4. **Consistent, repeatable and revisable P&A actions** All P&A actions modelled by the P-region are consistent in nature and are repeatable and revisable by designers and users;

5. **Personalisation induced from a core of hyperlink functionality** Within the P-region what is being tailored are unique, distinctive properties of hyperlink-based interaction, in contrast to emergent properties such as those that simply arise from the
coupling of database and user interface technologies in a particular manner using some particular protocol.

5.7 What Next?: A Model of System-Driven Personalisable Hyperlink-Based Interaction

In chapter 6, a model of system-initiated, personalisable hyperlink-based interaction is presented. This chapter extends the results of chapters 4 and 5 by proposing an architecture to model adaptivity, the A-region. Adaptivity is viewed as the process of allowing the system to take, by proxy, the initiative in personalisation actions described in chapter 5 in the light of the system’s inference of the information goals and the history of users. In summary, the A-region details how system-initiated tailoring, namely adaptivity, may be modelled as a coherent and consistent extension of the H- and P-regions.
Chapter 6
The A-Region: Future Work on Modelling System-Initiated Personalisable Hyperlink-Based Interaction

Chapter 5 described an abstract model of personalisable, non-adaptive hyperlink-based interaction. This chapter extends the results of Chapter 5 by proposing an architecture to model system-initiated, personalisable hyperlink-based interaction, the A-region. The formalisation of the A-region is the author’s future work.

The A-region details how system-initiated tailoring, namely adaptivity, may be modelled as a coherent and consistent extension of the H- and P-regions. Within the A-region adaptivity is modelled as the process of allowing the system to take the initiative in the personalisation actions described in chapter 5.

6.1 The A-Region: An Introduction

When adaptive functionality is layered over the H- and P-regions, both users and designers can define strategies as to when the system should take the initiative and actively tailor a user’s interaction in the light of that user’s information goals and history of use. The A-region therefore enables the modelling of adaptive hyperlink-based interaction. In this chapter, it is shown that system-initiated personalisation is, in principle, as expressive as user-initiated personalisation and requires no technologies other than those involved in user modelling and decision making from a user model. It is argued that in the model proposed, no additional concern is needed and that these technologies may be treated as “black boxes” that can be non-disruptively added to the model. This approach is viewed as beneficial because it enables appropriate user models and decision making algorithms, for particular application areas, to be included as and when appropriate implementation strategies require.
The chapter begins by describing the conceptual framework that underpins the A-region in section 6.2. Section 6.3 describes the additional dynamic behaviour of a HLBS when the A-region is layered over the P- and H-regions. In section 6.4, an architecture is proposed for an adaptive function. Broadly, this function enforces a decision theory when a change has been detected in a user’s recorded history and/or information goals. The process of enforcement is based, in part, on the current state of a user’s recorded history and information goals. Section 6.6 then details alternative strategies which may be used to extend the adaptive function proposed. Finally, a summary is given of the opportunities afforded by the A-region.

6.2 The A-Region: A Conceptual Framework

The concepts introduced by the A-region are now briefly described.

An information goal characterises what a user wishes to achieve in terms of information gathering when interacting with a HLBS. Broadly, information goals specify how the information domain should be tailored for a user.

For example, an information goal could be that a user is (or is not) interested in being presented with hyperpages that contain a particular phrase. Furthermore, an information goal may be a request to insert or delete an entire hyperpage or one of its component parts.

An information goal may be characterised as having two parts: the specification of its goal-scope, the semantics of which is the selection of a subset of hyperpages and a goal-action that the user wishes to be performed over the hyperpages in the goal-scope.

The goal-scope gs, is represented as a pair having the form \( \langle s, sc \rangle \) where \( s \) denotes the scope of the goal which may be an entire hyperpage or one of its component parts. For the purpose of this dissertation, a component part is assumed to be a chunk that may be referred to in the manner detailed in subsection 5.4.3 (e.g., ‘1, chunk’ refers to the first chunk in a hyperpage and ‘any, chunk’ refers to any chunk in a page). sc denotes a selection condition which those hyperpages that fall within \( s \) must satisfy. A selection condition is a primitive that specifies a regular expression and is true if, and only if, that regular expression occurs within the specified hyperpage or page part. It is assumed that there is also a vacuously true selection condition, denoted by true, that caters for the need to select an entire set of hyperpages. Let GS denote the set of all goal-scope pairs.

A goal-action ga, is represented as a triple having the form \( \langle a, as, asc \rangle \), where \( a \) denotes an action whose execution results in the tailoring of a subset of hyperpages, \( as \) denotes the scope of the action and \( asc \) denotes a selection condition which hyperpages that fall within the action’s scope must satisfy before the action is taken to tailor them.
Possible actions are: projections on sets of hyperpages (\texttt{drop-if} or \texttt{retain-if}) with which a set of hyperpages can be filtered using a selection condition and updates (\texttt{insert} and \texttt{delete}) with which hyperpages can be built and maintained. A scope may be any part of an entire hyperpage denoted by, \texttt{any}, or one of it’s component parts. Let $GA$ denote the set of all goal-scope triples.

Let $IG$ denote the set of all information goals. Therefore $IG = GS \times GA$. Thus an information goal $ig$, where $ig \in I$ is of the form $\langle gs, ga \rangle$ where $gs \in GS$ and $ga \in GA$.

In summary, a goal-scope may be read as “select a hyperpage from the set of all hyperpages only if $sc$ occurs in $s$”. A goal-action may be read as “execute action, $a$, if and only if $asc$ occurs in $as$”.

An information goal of the form,

\begin{quote}
“I am interested in being presented with hyperpages in which the phrase ‘year 1999’ occurs, keep these pages if they contain the phrase ‘year 2000 compliant’”
\end{quote}

using the model described above would be represented as:

\{<any, chunk, 'year 1999'>, <retain-if, any, chunk, 'year 2000 compliant'>\}

Where the goal-scope “I am interested in being presented with hyperpages in which the phrase ‘year 1999’” is represented by $\langle\text{any, chunk, 'year 1999'}\rangle$ and the goal-action, “keep these pages if they contain the phrase ‘year 2000 compliant’” is represented by the $\langle\text{retain-if, any, chunk, 'year 2000 compliant'}\rangle$.

A \texttt{goals-library} is defined to be a store of information goals.

From the description and model of an information goal described, it may be inferred that such goals could and perhaps should be modelled as personalisation requests as defined in section 5. This is a logical extension of the fact that a personalisation request is an expression of how a user wishes to tailor their hyper-network and this is also the purpose of a user specifying an information goal.

A \texttt{user’s history} characterises what is already known by the user about a hyperdocument and about HLBSs. This knowledge is based on a record of previous interactions with hyperdocuments and HLBSs. A history, $h$, is represented as a triple of the form $\langle e, t, d \rangle$ where $e$ denotes an event occurrence, $t$ denotes a time-point and $d$ denotes an event description. Let $H$ denote the set of all such triples.

A history entry may be read as “event $e$ occurred at $t$ and is described by $d$”. The form of $d$ will ultimately be determined by the nature of the events that the history is to record. The form of $t$ typically includes system clock ticks and the time of occurrence of real-time.
events (e.g., when a page was created or requested). A *history-library* is a store of user histories.

For the purposes of the model, it is assumed that the implementations of the goals-library and the history-library both provide the functionality of a modern database system. It is assumed that there exist mechanisms for scalable retrieval, associative querying, security of access, concurrency, versioning and transaction control. These two data stores may be viewed as a knowledge base containing a user’s model.

In the context of this dissertation, a *decision theory* is the background knowledge used by an inferencing mechanism capable of deriving implicit information by applying a set of active rules to extensionally stored data. More generally, a decision theory may be understood to be a general model of what is best to do and when. Central to the decision theory is a set of rules, each of which is an imperative command that is meant, in principle, to effect a state transition. Rules are said to be active, in that they express actions to be performed, dependent on context, as a reaction to events. In section 6.4, an architecture is presented in which active rules specify a function to model adaptivity.

An active rule $\rho$ may be characterised as a pair $\rho = <\kappa, \mu>$ [Fernandes et al., 1997]. $\kappa$, called the execution model of $\rho$, determines the behaviour of the execution conditions of $\mu$, the specification part of $\rho$. Broadly, the execution model specifies how and when the specification part $\mu$ will be applied.

The specification part $\mu$ is commonly referred to as an *ECA* (Event, Condition, Action) rule. Such expressions may be read as “when an event $E$ occurs and the condition $C$ holds in the current state of the underlying data, then perform action $A$". Each part of an *ECA* rule will be an expression in a specification language, therefore we may talk of an *event specification language ESL*, a *condition specification language CSL*, and an *action specification language ASL*. The semantics of a set of active rules will stem from the rules themselves and the execution model for applying them.

The application of active rules may be used to instantiate adaptive interaction between a HLBS and its users. Furthermore, such an approach allows for the enforcement of constraints on the state of the HLBS proper (e.g., rejecting a personalisation request if a user has a particular user model).

Implicit in the use of active rules as described above is the existence of a monitoring system whose role is to monitor the triggering of events. Such events may be primitive, in the sense that they correspond to an atomic occurrence of some phenomenon or may be composite events comprising several primitive events.

In summary, the conceptual framework of the A-region comprises a store of user information goals (goals-library), a store of user’s interactions (history-library) and a set of active rules. These active rules intuitively specify that on the occurrence of an event, if
some condition holds for some user, then execute some action that aims at making interaction more personal (e.g., suggest a personalisation request). A system for monitoring and detecting such events is outside the scope of this dissertation, however, such a system is assumed to be in place. A detailed discussion of active rules can be found in [Paton, 1999].

This section has described the concepts which underlie the conceptual framework of the A-region. In section 6.3, the additional dynamic behaviour of a HLBS when the A-region is layered over the P- and H-regions is described.

6.3 The A-Region: Dynamics of the A-Region

The additional dynamic behaviour of a HLBS when the A-region is layered over the P- and H-regions is now informally defined. Figure 6.1 is a simplified data flow diagram which expands the shaded oval in Figure 3.13. A-region functions do not conflict with P- and H-region functions and, indeed, rely on them being exactly as defined in Figure 5.1, except for the perform retrieval function that now handles more cases. This match is graphically represented by super-imposition, with the functions and flows that are identical in Figure 5.1 and Figure 6.1 drawn with thinner lines and named with a lighter typeface.

![Figure 6.1: Superimposing the A-Region onto the P- and H-Regions](image-url)
The architecture depicted in Figure 6.1 suggests one approach to add, in a non-disruptive manner, adaptive capabilities into the model of personalisable hyperlink-based interaction described in chapters 4 and 5.

The approach centres on an adaptation function. This function implements an inference engine over a decision theory, specified as a set of active rules, that describes which actions are more likely to yield the most benefits given some accumulated knowledge of past interactions. This accumulated knowledge is comprised of a goals-library, a history-library and the current state of the hyper-library database.

The operation of the adaptive function can be broadly understood as enforcing the decision theory when an event is detected, taking into account the state of the history-library. The actions which the inference engine is in charge of suggesting are personalisation actions as defined in the grammar for personalisation requests (see section 5.4.3).

The adaptive process starts when an event is detected by some monitoring system. In the model such an event is assumed to be a change in the state of the history-library as a result of some real-time event such as a request for a hyperpage or the insertion of an information goal into the goals-library. As a result of the detection of an event, the adaptive function consults the goals-library. This consultation process allows the adaptive process to determine which entries have been made in the goals-library and therefore its current state.

Using this knowledge the adaptive function acts as an inference engine deriving implicit information by applying a set of active rules to the current database state. Broadly, the adaptive function applies the set of active rules to each goal in the goals-library. If the condition part of an active rule is satisfied it triggers a call to an action to be taken. The condition passes as parameters, to this action, the goal that was the subject of the condition.

This information is then used to construct and then suggest to the user a well formed personalisation request that is aimed at making user interaction more personal. If this suggested personalisation request is accepted by the user then it is submitted to and acted upon by the functions that comprise the P-Region as defined in chapter 5. Note, it is also possible for the adaptive function not to suggest to the user a particular personalisation request, but instead directly submit this personalisation request directly to the functions that comprise the P-region.

In summary, the addition of the A-region allows the HLBS to take the initiative and launch personalisation requests on behalf of a user. For this to be possible, the adaptive function draws upon a model of a user’s goals (both explicitly stated and inferred), a history of a user’s requests and a decision theory. Such a decision theory will embody strategies towards achieving information goals. The architecture depicted in figure 6.1 shows that these components, using any implementation, may be added to the model in
a non-disruptive manner. This approach allows these components to be viewed as black boxes, therefore, in particular application domains appropriate user models and decision theories may be implemented.

This subsection has described the dynamic behaviour of a HLBS instantiating the A-region. In the following section, an architecture is proposed for the implementation of an adaptive function that forms, in part, future research to be conducted. The purpose of this architecture is to illustrate the lines upon which such a function may be seamlessly incorporated into the model proposed.

6.4 A Proposed Architecture for the Adaptive Function

This section details an architecture for an adaptive function. There are three distinguishing components in the A-Region:

1. *An active rule set* that express actions to be performed, dependent on context, as a reaction to events;

2. A *knowledge base* comprised of a goals-library and a history-library;

3. *An active database of hyperpages and hyperpage annotations* (i.e., the hyper-library) that are the subject of the adaptation process.

Each of these components are now described in detail.

6.4.1 An Active Rule Program

A rule, denoted by $ECA$, is assumed to have the following (Event, Condition, Action) form:

\[
on \langle event \rangle \text{ if } \langle condition \rangle \text{ then } \langle action \rangle .\]

Informally, an event may range over external and/or internal events. An event may be atomic or *composite*. A composite event is built up from one or more atomic events. However, in the context of this dissertation events are assumed to be primitive in nature. The event itself is an expression in a specification language, *ESL* that returns either a Boolean value or a set of values that indicate what triggered the event. For the purpose of the model, a simple *ESL* is assumed within which one may only express the detection of a change in the state of the history-library, denoted by $\text{Detect}$ (history-library).

A *condition* is an expression written in a specification language, *CSL*. In the model, a simple *CSL* is assumed in which conditions have the form $X$, where $X$ is a Boolean
value. The evaluation of a condition may involve parameters passed in as the result of the
detection of an event. When a condition is evaluated as \textit{true} it triggers a call to an action
to be taken and in the case of \textit{false} does not. In the model, a condition is assumed to
return a set of variable bindings that denote the user goal which was emulated using the
condition. Each value binding is associated with a template variable. A template variable
is a place holder for a value returned from the evaluation of a condition.

In principle, an \textit{action} is an expression in a specification language, \textit{ASL}. In the context
of this dissertation, an action expression takes the form of a personalisation request in-
terspersed with template variables. If a condition expression specified by a rule has been
satisfied, the occurrence of template variables within an action expression signals that the
execution of textual operations need to be carried out to obtain the value assigned to the
template variable. The result of the execution of these textual operations is a well formed
personalisation request. When an action expression is interpreted, it results in a call to
the function \textit{personalise}. This call has as its parameter the well formed personalisation
request.

All three specification languages, ESL, CSL and ASL, are dependent on the definition
language of the data model underlying a given database for their lexicon.

Recall, an active rule \( \rho \) is a pair, \( \rho = < \kappa, \mu > \). \( \kappa \) called the execution model of \( \rho \) determines
the behaviour of the execution conditions of \( \mu \), the specification part (rule) of \( \rho \). The
execution model specifies how and when the specification part \( \mu \) will be applied. The
true semantics of a rule base stem both from the rules themselves and from the execution
model for applying them. Consideration of the relationship between the triggering of an
event and the rule execution leads to several possible approaches to processing ECA rules.
These possibilities include the following:

Assume that there is only one user of a system, then suppose that user issues a transaction
\( t = c_1 \ldots c_n \) where each \( c_i \) is an atomic command, then in a traditional database system
application of \( t \) will yield the sequence \( I_0, I_1 \ldots I_n \) of database states, where \( I_0 \) is the
original state and where each \( I_i + 1 \) is the result of applying \( c_i + 1 \) to the state \( I_i \). In an active
database, \( \kappa \) may be used to specify that upon the triggering of an event, the corresponding
rule is processed using an:

1. \textit{Immediate Approach} The rule is fired immediately on detection of the event in the
main transaction. The execution sequence of the main transaction is then suspended
until after any actions resulting from evaluating the rule are completed. If rules are
nested, then the action part of a rule triggers the firing of a further rule, the first
rule is suspended until the second rule is completed and so on. The semantics of
immediate firing may be broadly understood as follows:

Assume that the event of each rule is \textit{true} and that a rule, \( r \) with action \( d_1 \ldots d_m \) is
triggered (i.e., its condition is true) in the state \( I_1 \) of the previous sequence of states.
Then the sequence of database states might start with $I_0, I_1, I'_1, I_2, I'_2, \ldots, I'_m, \ldots$, where $I'_{j+1}$ is the result of applying $d_{j+1}$ to $I'_{j}$. After $I'_m$, the command $c_2$ would be applied;

2. **Deferred Approach** The rule is fired and executed as the final part of the transaction before it commits. As with the immediate approach, rules are executed as part of the main transaction. If rules are nested, then further rules may be executed immediately or deferred until completion of the triggering rule. The semantics of deferred firing may be broadly understood as follows:

Assume that the event of each rule is `true`, and that a rule, $r$ with action $d_1 \ldots d_m$ is triggered (i.e., its condition is true) in the state $I_1$ of the previous sequence of states. The deferred firing gives rise to a sequence of states $I_{orig}, I_{user}, I_2, I_3, \ldots, I_{curr}$, where now $I_{orig}$ is the original state, $I_{user}$ is the result of applying the user-requested transaction, and the states $I_2, I_3, \ldots, I_{curr}$ are the results of applying the actions of fired rules. The sequence shown may be extended if additional rules are to be fired.

A third approach is current rule execution where the rule is executed in a separate transaction initiated by the triggering event. The rule transaction executes concurrently with the main (parent) transaction. Such an approach requires the use of conventional concurrency control mechanisms.

It should be clear from this brief discussion of execution models that many intricacies may arise. These are not discussed here, but may be found in texts such as [Abiteboul et al., 1995, Paton, 1999]. The specification of execution rules fall outside the scope of this dissertation. However, for the purpose of defining the adaptive function, the execution of rules is assumed to be immediate.

### 6.4.2 A Knowledge Base

This subsection defines a syntactic characterisation of a simple knowledge base. The knowledge base is comprised of a goals-library and a history-library. It is assumed that the knowledge base is accessed by a single user. No attempt is made to model the functionality required for multi-user access, other than to specify that a user has a reference that may be used to identify a subset of user goals that belong to a user.

A *goals-library* is a set of information goals, denoted by $IG$ (see subsection 6.2). A *history-library* is a set of recorded user interactions, denoted by $H$.

A knowledge base is defined as the set $IG \cup H$, denoted by $KB$. 

151
6.4.3 An Active Database

This subsection defines a syntactic characterisation of an active database component for an adaptive function.

Let $HLDB$ denote a database containing hyperpages and hyperpage annotations (i.e., the hyper-library) whose data definition language, DDL, data manipulation language, DML and data query language, DQL are respectively $D$, $M$, $Q$. Similarly, let $E$, $C$, $A$ denote the event, condition and action specification languages whose lexicon is induced from $D$.

Furthermore, let $C \subseteq Q$ and $A \subseteq M$. An $ECA$ rule set $HLDB^+$ over $HLDB$ is a set of expressions of the form $E : C \rightarrow A$ such that $E \in E$, $C \in C$ and $A \in A$. The active rule program, $HLDB^*$ over $HLDB \cup KB$ is a set of pairs $\rho = \langle \kappa, \mu \rangle$ where $\kappa$ is a set of parameters for the execution of $\mu$ and $\mu \in HLDB^+$.

The active database of the adaptive function is then defined as a pair, $\langle HLDB \cup KB, HLDB^* \rangle$.

6.5 Example Active Rules

This section presents several examples of active rules which may be used in conjunction with a user’s information goals and history to infer personalisation requests.

6.5.1 Assumptions and Notational Conventions

Assume the existence of a function to monitor events, denoted by $\text{Detect}(\text{history-library})$. The role of this function is to identify any change in the underlying state of the history-library. Given a history library, if a change is detected, the function $\text{detect}$ returns the Boolean value $true$, else it returns $false$.

Assume the existence of a function $\text{match}$ whose role is to determine if a particular information goal has been specified by a user and recorded in the goals-library or that a particular event has been logged in the history-library. Given an information goal and a goals-library, the function $\text{match}$ returns a set of variable bindings whose values denote the goal that was matched. Similarly, given a set of triples that denote a user’s interaction history and a history-library, the function $\text{match}$ returns a set of variable bindings whose values denote the history entries that were matched.

Assume the existence of a procedure to display a text string, denoted by $\text{Display}(s)$ where $s$ is a text string representation of a personalisation request. The role of this procedure is to display to the user a personalisation request that the adaptive function has suggested would meet that user’s information goals.
Assume the existence of a function **Submit**. The role of this function is to submit a well-formed personalisation request to the function *personalise* described in chapter 5.
Let $IG$ be a goals-library comprised of the following pairs

\[
\{(\text{'any,chunk'}, \text{'year 1999'}), \text{<retain-if,'any,chunk', 'Year 2000'>} ), \\
(\text{'true'}, \text{null}, \text{<delete,'1, chunk',null}>), \\
(\text{'1, chunk'}, \text{'Goldsmiths'}, \text{<insert,'1, chunk', 'declassify'>}), \\
(\text{'1, chunk'}, \text{'work in progress'}, \text{<drop-if,'1, chunk', 'Year 1999'>}) \\
\}
\]

Let $H$ be a history-library comprised of the following triples

\[
\{\text{'event-1',00:10,'goals-library-change'}, \\
\text{'event-2',00:20,'hyperpage-request'}, \\
\text{'event-3',00:25,'goals-library-change'}, \\
\text{'event-4',00:40,'goals-library-change'}, \\
\text{'event-5',00:60,'goals-library-change'} \\
\}
\]

Let $HLDB$ denote a database representation of the hyper-library. Then let $HLDB^*$ denote an active rule program over $HLDB$. A possible active rule set of $HLDB^*$ is detailed in figure 6.2.

### 6.5.2 Application of Active Rule Set

On the detection of a change in the history-library, the condition part of a rule is tested. This test aims to deduce if one or more entries in the goal-library has a goals-scope $gs$, whose values of $s$ and $sc$ match and a goal-action $ga$, whose values for $a$, $as$ and $ga$ also match. If one or more such rules exist then each is processed\(^1\).

For each information goal that matches, the value bindings for its $gs$ and $ga$ are returned from the condition and are passed as parameters to the action part of the rule which is then triggered. Each value binding is associated with a template variable (place holder for a value returned) found in the action part of the rule.

The action part of a rule is an expression that takes the form of a personalisation request inter-dispersed with template variables. Upon the triggering of the action, a call is made to the function $\text{bind}$ that carries out a sequence of textual operations needed to obtain the value assigned to the template variable dispersed within it. $\text{bind}$ returns a well formed personalisation request. This request is then presented to the user so that they may decide to issue the personalisation request or not. If confirmed by the user, the function $\text{Submit}$

\(^1\)The processing of rules within the goals-library is assumed to be sequential, starting from the first recorded goal that matches and ending with the last.
On % Rule 1
detect(history-library)
If
match(goals-library, (\(<s, sc>, \langle\text{\textquote{retain-if}}, as, asc\rangle\))
Then
Submit(Display(bind('select-page-if
    [s] contains sc
    hp-then-do {
        retain-if [as] contains asc
    }
    )
)

On % Rule 2
detect(history-library)
If
match(goals-library, (\(<\text{\textquote{true}}, null>\), \(<\text{\textquote{delete}}, as, null\rangle\))
Then
Submit(Display(bind('select-page-if
    true
    hp-then-do {
        delete, [as]
    }
    )
)

On % Rule 3
detect(history-library)
If
match(goals-library, (\(<s, sc>, \langle\text{\textquote{insert}}, as, asc\rangle\))
Then
Submit(Display(bind('select-page-if
    [s] contains sc
    hp-then-do {
        insert [as]
            content { X:= asc }
            rendering { X }
    }
    )
)

On % Rule 4
detect(history-library)
If
match(goals-library, (\(<s, sc>, \langle\text{\textquote{drop-if}}, as, asc\rangle\))
Then
Submit(Display(bind('select-page-if
    [s] contains sc
    hp-then-do {
        drop-if [as] contains asc
    }
    )
)
is executed. This function submits the personalisation request to the function \textit{personalise} which then processes it in the manner described in chapter 5.

\textit{Rule 1} caters for the situation where a user wishes to express a goal of the form, “I am interested in all hyperpages that meet the selection condition \textit{sc}. From this subset of hyperpages I am only interested in being presented with those pages that meet the selection condition \textit{asc}”.

A test is made that aims to deduce if one or more entries in the goals-library has a goal-scope \textit{gs}, whose values of \textit{s} and \textit{sc} are left unspecified and a goal-action \textit{ga}, whose value for \textit{a} is the token \texttt{retain-if} with the remaining values for elements in \textit{ga} being left specified.

\textit{Rule 2} caters for the situation where a user wishes to express a goal of the form, “I wish to delete the page part \textit{as}, from all hyperpages in the hyper-library”.

\textit{Rule 3} caters for the situation where a user wishes to express a goal of the form, “I am interested in all hyperpages that meet \textit{sc}. For this subset of hyperpages I wish to insert into the page part \textit{as}, the expression \textit{asc}.

\textit{Rule 4} caters for the situation where a user wishes to express a goal of the form, “I am interested in all hyperpages that meet the section condition \textit{sc}. From this subset of hyperpages I am only interested in being presented with those pages that do not meet the selection condition \textit{asc}”.

\section{6.6 Extending the Adaptive Function}

The formal definition of an active rule set, knowledge base and active database representation of a hyper-library given in section 6.4 comprise the core elements of one approach to constructing an adaptive function.

Section 6.5 then presented several examples of active rules which may be used in conjunction with a user’s information goals and history to infer personalisation requests. The interpretation of adaptivity given here has been purposefully limited. It aims to illustrate how potential strategies for implementing adaptivity may be realised. This section details alternative strategies which may be used to extend the adaptive function proposed.

It is possible to endow an adaptive function with functionality to enable it to infer a personalisation request as a result of a user request for a hyperpage. Such an approach would involve the adaptive function detecting that a user has issued a request for a hyperpage (as opposed to a change in the state of the hyper-library) and then inferring an appropriate personalisation request by applying a set of active rules and consulting the current state of that user’s information goals and history of interactions.
Further functionality could be provided by enabling the adaptive function to issue personalisation requests based on only: user goals, user histories or requests for hyperpages. This approach would enable the issuing of personalisation requests to be studied with a finer level of granularity.

What may be concluded is that different adaptive functions may be required and as a result, an architecture such as the one described in this chapter is suitable for including alternative adaptive functions. Furthermore, it has been shown that no further functionality is required other than that of a user model and a decision making algorithm and that these components may be treated as black boxes. The only requirement of such a black box is that it possesses the functionality to issue a personalisation request as defined by the P-region. What mechanisms are used to determine which personalisation request to issue is context dependent and therefore, at the level of abstraction at which this model is cast, need not be further specified.

6.7 Opportunities Afforded by the A-Region

The abstract, additive model of system-initiated, personalisable hyperlink-based interaction described in this chapter is one answer to the question of how system-initiated P&A actions could be made available to users. It qualifies as one such answer insofar as it indicates how, given a conceptual framework such as those that underlie the H- and P-regions, it is possible to devise a simple architecture to model system-initiated tailoring, adaptivity. Adaptivity in the proposed model is viewed as the process of allowing the system to take the initiative in issuing personalisation requests induced from the formal definition of what it is that we wish to personalise, namely, the emergent properties of HLBSs.

It is complete, in the sense that it is expressive enough to illustrate how adaptivity can be modelled as a coherent and consistent extension of the H- and P-regions. Furthermore, it shows that using the conceptual frameworks of the H- and P-regions, adaptivity can be as expressive as personalisation. This is an important result, insofar as that, to the author’s knowledge, no system proposed has been able to implement this level of functionality and no model has shown this to be possible.

Attention is drawn to the fact that adaptivity is rooted in the implementation issues associated with the construction of a user model and decision making algorithm. As such, although these components are required, their scope, form and levels of functionality should be modelled for particular application areas, as and when appropriate implementation strategies are required.

In section 6.4 an architecture is proposed for an adaptive function, the major components of which are: a database of hyperpages and hyperpage annotations, $HLDB$, a knowledge base, $KB$; an active rule program, $HLDB^*$ over $HLDB \cup KB$; and a monitoring system.
for detecting events. It is shown that the active rule program can be extended in many ways to accommodate rules written using any appropriate ESL, CSL, and ASL. The choice of these will be ultimately determined by the underlying data model used in the construction of the database of hyperpages and hyperpage annotations. The central component of the adaptive function is an active database, defined as a pair $\langle HLDB \cup KB, HLDB^* \rangle$, where $HLDB$ is a database of hyperpages and hyperpage annotations, $KB$ is a knowledge base and $HLDB^*$ is an active rule program.

Attention is drawn to the fact that even as the detail of active database component proposed could be presented in equivalent ways, the methodological procedure of defining its form, modelling it formally and then constructing from that formalisation an adaptive function is a contribution that can be made use of in other settings, under different assumptions and using alternative conceptualisations of a user model and decision making algorithm.

In summary, the abstract model of system-initiated, personalisable hyperlink-based interaction contributed by this chapter possesses the following beneficial characteristics:

1. *Adaptivity is induced from a formal definition of personalisation* Within the A-region, the system tailors interaction by issuing personalisation requests as defined by the P-region that in turn have been induced from a core of hyperlink functionality, the H-region;

2. *Adaptivity is as expressive as personalisation* All decisions designers and users of hyperdocuments make with regard to content, structure and presentation of units of information are able to be the subject of P&A actions issued by the system;

3. *Adaptivity can and should be modelled separately from personalisation* Within the A-region, adaptivity is modelled separately from personalisation. This approach allows a simple representation of what adaptivity is and how in particular implementations appropriate user models and decision making algorithms can be used to implement P&A actions;

4. *Adaptivity is system-oriented and not HLBS specific* The notion of adaptivity is shown to be system-oriented, it is rooted in a mechanism that allows the system to detect events, make conjectures and issue commands. It is not specific to HLBSs but can be modelled, for example, using established active database theories and implemented using established database management systems. The functions used to detect events, make conjectures and issue commands may be modelled and then implemented in many ways. This conclusion lends weight to the view that such functions may be viewed as black boxes whose composition is implementation specific.
6.8 What Next?: A PA-HLBS to Perform Empirical Studies of Personalisable, Adaptive Hyperlink-Based Interaction

In chapters 4, 5 and 6, a model of personalisable, adaptive hyperlink-based interaction is proposed. The model is an abstract model, as many steps removed from concrete implementations as necessary to allow a systematic, exhaustive investigation of P&A issues in HLBSs. The model proposed provides answers to two fundamental questions proposed in 1.2.1, namely, what is the scope of P&A in HLBSs and which P&A actions could be made available to users? The remaining question, which P&A actions should be made available to users? requires empirical studies conditional on the answers addressed by the research reported here.

To perform these empirical studies, it is necessary to construct a HLBS that embodies the main concepts underlying the model proposed in chapters 4 to 6. Although such empirical studies are outside the scope of the research reported in this dissertation, a personalisable, HLBS (PAS) has been developed for this purpose. The goals, design and implementation of PAS are detailed in chapter 7.
The purpose of this chapter is to substantiate the claim that the formal, abstract approach to modelling personalisable, adaptive hyperlink-based interaction taken in this dissertation not only allows for a greater understanding of what personalisation and adaptivity means, but may also be an aid in the design of such systems.

This chapter describes the development of a personalisable HLBS, PAS, that embodies the main concepts underlying the abstract model of personalisable, adaptive hyperlink-based interaction described in chapters 4 to 6. The aim of PAS is to allow for a principled, systematic empirical study to be carried out into the effects of P&A actions. Note, the PAS system is only a “proof of concept” and is intended as a test-bed and demonstrator for the results reported in this dissertation.

The PAS consists of two major components: an instantiation of the H-region as a WWW-based application for realising dynamically generated hyperdocuments; and an instantiation of the P-region as a parser for personalisation requests.

Parsing is the process used to describe the analysis of the structure of expressions of a language. Parsing consists of lexical analysis, syntactic analysis and semantic analysis. Lexical analysis is the grouping of individual characters of a source language program into tokens. Syntactic analysis is the parsing of the phrase structure of the source language program to determine whether the stream of tokens from the lexical analyser forms a valid sentence in the source language grammar. The output is a syntax tree. Semantic analysis is the assignment of meaning. This is achieved by mapping the syntax tree into a structure in the target language, of which the audience (or machine) is assumed to know the meaning. The assignment of meaning to a personalisation request is formally described in subsection 5.5.3.

The semantics of personalisation requests are implemented by a semantic analyser whose input is an abstract syntax tree representation of a personalisation request outputed by the syntactic analysis phase of the parsing process. The role of the semantic analyser is to generate for a given personalisation request its corresponding representation as a
sequence of Standard Query Language (SQL) expressions\[1\] [Abiteboul et al., 1995]. Data structures for the storage and management of hyperpages and hyperpage annotations have been implemented using the relational data model and these form the subject of personalisation requests issued as SQL expressions.

This chapter begins by presenting a brief overview of the design goals and implementation strategy taken in the development of PAS. Following this, an architectural overview of the PAS is presented and its main components described. The chapter then turns its attention to the development tools, techniques and platform used in the construction of PAS. These are briefly described and future work on the PAS is outlined.

7.1 PAS: An Overview

The PAS is a personalisable HLBS developed using the formal approach described in chapters 4 to 6. The PAS adopts the strategy of separating out into different components the need to support: the functionality afforded by the formal specification language for personalisation requests (back-end component) and; a dynamic style of WWW-based hyperlink interaction (front-end component).

The PAS brings together into as tight a coupling as possible the programming paradigms of logic-based programming (applied to the development of a parser) and event-driven database programming (used in the development of the dynamic WWW-based application). This tight coupling is pitched so that either paradigm is not snapped by attempts to subsume the other.

The technical goals of the PAS are:

1. To implement a lexical analyser for the formal language specifications for personalisation requests, hyperpage and hyperpage annotations;
2. To implement a syntactic analyser for the formal language specifications for personalisation requests, hyperpage and hyperpage annotations;
3. To implement a semantic analyser to translate a valid personalisation request into a sequence of SQL statements that represent the intended meaning of the given personalisation request;
4. To implement a hyper-library as a database (using a relational database management system) comprised of relations over the relational schemas for hyperpages and hyperpage annotations as described in section 5.5.1;

\[1\]SQL is a classical approach to implementing relational algebraic expressions, over a database composed of relations defined using the relational data model.
5. To implement a personalisation program that given a SQL statement representation of a personalisation request, executes that SQL statement over a hyper-library database, thereby realising a state transition that is the intended meaning of the personalisation request;

6. To implement a hyperpage generator program that given a request for a hyperpage, dynamically retrieves content, mark-ups for that hyperpage and then composes this content and mark-ups into a rendering expression;

7. To implement an apply preference program that for a given hyperpage retrieves the associated annotation (if one is present) and a rendering expression, performs any post composition rewriting specified by that annotation over the rendering expression and then returns it to a WWW browser (UIS).

The purpose of PAS is therefore not to implement the functions of the H- and P-regions exactly as described in chapters 4 to 6, but to take these as a template for the structural design and subsequent implementation of a system that embodied the functionality described in these chapters. In the following section an architecture of PAS is presented and its main components described.

### 7.2 The Architecture of PAS

The architecture of PAS is partitioned into a back-end component and a front-end component. Given a text file containing a set of personalisation requests, the back-end component performs: lexical analysis; syntactic analysis; and semantic analysis (the generation of SQL statement representations of valid personalisation requests) on this set of personalisation requests. The front-end component is responsible for executing SQL statement representations of valid personalisation requests over a database containing hyperpages and hyperpage annotations. The front-end is also responsible for: dynamically retrieving content, mark-ups and hyperpage annotations from the database; dynamically composing that content into a rendering expression; applying post composition rewrites to rendering expressions; and returning rendering expressions to a WWW browser (UIS). As such, the front-end is the active component of PAS and the back-end is the stable component. Figure 7.1 depicts the architecture of PAS. Some comments on Figure 7.1 are now provided.

Formal language specifications of personalisation requests, hyperpage and hyperpage annotations are stored in text files that may be submitted to the lexical analyser, read_tokens. After analysing the structure of these source files, read_tokens passes on to a syntactic analyser, Phrase, a stream of tokens.
Figure 7.1: Architecture of PAS
Phrase reads in a stream of tokens and with reference to the formal grammars for personalisation requests, hyperpages and hyperpage annotations, determines if this stream of tokens forms a valid sentence. If a sentence is valid then Phrase unambiguously derives its parse tree.

In the case of a personalisation request, the parse tree is passed onto a semantic analyser, aphlbs_semantics. This analyser generates for a given personalisation request parse tree its corresponding representation as a sequence of SQL statements. These statements are then stored in a text file.

To implement the WWW application, a decision was made to implement hyperpage and hyperpage annotation specifications as relations over the relational schemas defined in section 5.5.1. The motivation for this was based on the relatively high level of functionality SQL provides for querying data stored using the relational model. Although it is possible to implement selective data retrieval from text files, such an approach would have to require a higher level of programming effort and would introduce the need for complex programs to analyse the structural aspects of these text files.

Recall that the transformation of hyperpage and hyperpage annotations specification text files into relation schemas is achieved by retrieving, in sequence, each hyperpage (or hyperpage annotation) and then analysing its structure. This analysis involves reading each hyperpage from top to bottom and then transforming each part of its structure (i.e., entry-points, C-specs, R-specs and exit-points) into appropriate relations. When reconstructed as relations each part of the structure of a hyperpage (or hyperpage annotation) is given an extra attribute value as a unique identifier. This identifier acts in the role of the primary key for that relation. Identifiers are created in the same sequence as the transformation of hyperpages (or hyperpage annotations). As a side effect these identifiers also hold the ordering in which page parts were transformed and stored into their relations. The transformation process is assumed to be implementation specific, however, for the implementation of the PAS this was achieved manually.

The personalisation program retrieves SQL statements stored by the semantic analyser and executes these over the relations for hyperpages and hyperpage annotations. Such execution results in a state transition of the database, that is the intended meaning of the personalisation request.

On receiving a request for a hyperpage, the generate hyperpage program issues a sequence of SQL queries to retrieve that hyperpage’s content and mark-ups. The hyperpage generation program then composes the hyperpage into a rendering expression. This rendering expression is then passed onto the apply preferences program.

The apply preferences program retrieves a hyperpage annotation for a given hyperpage if one exists. The program then applies any post composition rewrites to rendering expressions specified by the hyperpage’s annotation. Following this, the apply preferences
program passes the rewritten rendering expression on to the UIS (WWW browser) that issued the request to retrieve a hyperpage. The UIS then renders the rendering expression and presents it as a personalised hyperpage to a user.

This section has described an overview of the architecture of PAS. In the following section the process of parsing a personalisation request (the back-end component) is described in detail.

7.3 The Back-End Component of PAS

The back-end component of PAS is responsible for parsing a personalisation request. Parsing is the process used to describe the analysis of the structure of expressions of a language. Parsing consists of lexical analysis, syntactic analysis and semantic analysis. The following section describes the implementation of the processes of lexical analysis, syntactic analysis and semantic analysis of personalisation requests as defined in 5.4.3. The parser has been implemented using the programming language Prolog [Clocksin and Mellish, 1994] and can be found in appendix D.

7.3.1 Lexical Analysis of Personalisation Requests

A lexical analyser takes as input a source language and produces as output a stream of tokens for a syntax analyser. Tokens are instances of general concepts such as integers and reserved words of the language. In carrying out lexical analysis, the lexical analyser reads a representation (lexemes) of these tokens. For some tokens, such as those associated with a reserved word, there will be only a single possible lexeme. A lexeme can be thought of as an attribute of the token. For example, the lexeme associated with an integer is a number and the lexeme associated with a text string is a sequence of characters. Tokens, such as reserved words, with a single lexeme need no explicit representation. The lexical analyser may also be used to carry out administrative tasks such as stripping out comments and white spaces between tokens.

The function \texttt{read\_tokens}

The process of lexical analysis has been implemented as a function, \texttt{read\_tokens}. The function \texttt{read\_tokens} takes as input a text file containing a personalisation request and produces as output a comma separated stream of tokens.

The signature of the function \texttt{read\_tokens} is

\[
\texttt{read\_tokens : text\_file \rightarrow token\_stream}
\]
Figures 7.2 and 7.3 show examples of the use of the function \textit{read\_tokens}.

Given a text file containing the personalisation request

\begin{verbatim}
select_page_if
    page contains '1997'
hp_then_do{
    delete[2,chunk]
}

delete[2,chunk],
\end{verbatim}

the function \textit{read\_tokens} yields the following token stream

\begin{verbatim}
select_page_if,page,contains,1997,hp_then_do,
'{',delete,'[',2,',',chunk,']','}',,'.'
\end{verbatim}

---

Figure 7.2: Personalisation Request and Generated Token Stream: Example 1

Given a text file containing the personalisation request

\begin{verbatim}
select_page_if
    true
hp_then_do{
    retain_if [2, chunk] contains '1998'
}

delete[2,chunk],
\end{verbatim}

the function \textit{read\_tokens} yields the following token stream

\begin{verbatim}
select_page_if,true,hp_then_do,' 
'{',retain_if,'[',2,',',chunk,']','}',contains,1998,'}','.'
\end{verbatim}

---

Figure 7.3: Personalisation Request and Generated Token Stream: Example 2

\subsection{Syntactic Analysis of Personalisation Requests}

A syntax analyser takes as input a stream of tokens from the lexical analyser. The role of the syntax analyser is to determine if this stream of tokens form a valid sentence in the language's grammar. If a sentence is valid then it is possible to unambiguously derive its parse tree. A successful parse of a token stream will establish the structure of the sentence in terms of the grammar. The parse tree shows how symbols link together into phrases, how the phrases form larger phrases and ultimately, how the phrases link together to form a sentence.

Technically, a parse tree has exactly one leaf for each token of the input and one internal node for each grammar rule derived during the parse. Such a parse tree is referred to as a concrete parse tree representing the concrete syntax of the source language. A concrete
parse tree may contain punctuation tokens that are redundant and convey no information. Such tokens may have been useful in providing semantics to the input string, but once the parse tree is built, the structure of the tree conveys this structuring information more conveniently. The abstract syntax tree conveys the phrase structure of the source program, with all parsing issues resolved but without any semantic interpretation.

The function \textit{Phrase}

The process of syntax analysis has been implemented as a function, \textit{Phrase}. The function \textit{Phrase} takes in as input a token stream and produces as output an abstract syntax tree.

The signature of the function \textit{Phrase} is

\[
\text{Phrase} : \text{token	extunderscore stream} \rightarrow \text{abstract	extunderscore syntax	extunderscore tree}
\]

Figures 7.4 and 7.6 show example token streams. The corresponding abstract syntax trees are detailed in Figures 7.5 and 7.7.

\begin{verbatim}
select_page_if, page, contains, 1997,
hp_then_do, '{', delete, '[', 2, ',', 'chunk', ']', '}', '.
\end{verbatim}

Figure 7.4: Token Stream for Example 1

\begin{verbatim}
select_page_if, page, contains, 1997,
hp_then_do, '{', delete, '[', 2, ',', 'chunk', ']', '}', '.
\end{verbatim}

Figure 7.5: Abstract Syntax Tree of Token Stream: Example 1
select_page_if, true, hp_then_do,
'{}, retain_if, '[', 2, ',', 'chunk, '], contains, 1998, '}', '.

Figure 7.6: Token Stream for Example 2

Figure 7.7: Abstract Syntax Tree of Token Stream: Example 2
Term Representation of Abstract Syntax

For convenience, an abstract syntax tree may be represented as a term. Figure 7.8 shows a term representation of the abstract syntax tree shown in Figure 7.5. Figure 7.9 shows a term representation of the abstract syntax tree shown in Figure 7.7.

```
personalisation_request('select_page_if',
    condition(atomic_condition(
        atomic_condition_on_page('page',
            containment_expression('contains', 1997))),
    action_list(action('hp_then_do',
        hp_update(hp_structure_update('delete',
            page_part(relevant_chunk(shift(signed_integer(+, 2)), 'chunk'))))))
)
```

Figure 7.8: Term Representation of Abstract Syntax Tree: Example 1

```
personalisation_request('select_page_if',
    condition(atomic_condition('true')),
    action_list(action('hp_then_do',
        hp_update(hp_structure_update(projection_operation('retain_if'),
            condition_on_hp_construct(atomic_condition_on_hp_construct(
                page_part(relevant_chunk(shift(signed_integer(+, 2)), 'chunk'))),
                containment_expression('contains', 1998))))))
)
```

Figure 7.9: Term Representation of Abstract Syntax Tree: Example 2

### 7.3.3 Semantic Analysis of Personalisation Requests

The remaining step is to assign meaning to the source language program (personalisation request). This is achieved by mapping the abstract syntax into a formalism (target language) of which the target audience (or machine) knows the meaning. This process has been formally described in subsection 5.5.3. For the purpose of implementing the semantics of personalisation requests, the relational algebraic statements described in section 5.5.3 are translated into SQL statements. These statements are then stored in a text file so that subsequently, they may be referenced by the personalisation program that executes them over the database of hyperpages and hyperpage annotations to realise a personalisation request.

The relational schemas for the hyper-library are reproduced here for the convenience of the reader. Recall that relation names are denoted by a sequence of upper-case **sans-serif** letters, the corresponding name in lower-case **sans-serif** letters denotes an identifier of an entity modelled by the relation. The primary key of each relation is underlined.
The function \textit{aphlbs\_semantics}

The process of semantic analysis has been implemented as a function, \textit{aphlbs\_semantics}. The function \textit{aphlbs\_semantics} takes in as input a term representation of an abstract syntax tree for a personalisation request and returns a sequence of SQL statements that are stored in a text file.

The signature of the function \textit{aphlbs\_semantics} is

\[ \text{Phrase} : \text{abstract\_syntax\_tree} \rightarrow \text{SQL\_statements} \]

Figures 7.4 and 7.6 show example term representations. The corresponding SQL statement representations are detailed in Figures 7.11 and 7.12.

The SQL statements shown in figures 7.11 and 7.12 represent the intended meaning of the personalisation requests shown in figures 7.4 and 7.6. SQL statements generated as a result of the semantic analyser, \textit{aphlbs\_semantics} are written to a text file. This text file is subsequently read by the personalisation program.

Figure 7.13 depicts the back-end component of PAS as a fragment of a Prolog [Clocksin and Mellish, 1994] program code. The program is used to implement the lexical and syntactic analysis of hyperpages, hyperpage annotations and personalisation requests. The program also implements the semantic analysis of personalisation requests.

This section has described the process of parsing a personalisation request that forms the back-end component of the PAS. In the following section, the processes of executing SQL statement representations of personalisation requests and the dynamic generation of personalised hyperpages is described. These processes are the responsibility of the front-end component of PAS.
INSERT PAGE.page, PAGE.chunk, PAGE.shift INTO SELECTEDPAGES
WHERE PAGE.page IN (SELECT PAGE.page FROM PAGE)
AND (PAGE.page IN
% select page if
(SELECT PAGE.page FROM PAGE
WHERE PAGE.chunk IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.entrypointset IN
(SELECT ENTRYPINTSET.entrypointset FROM ENTRYPINTSET
WHERE ENTRYPINTSET.entrypoint IN
(SELECT ENTRYPINT.entrypoint FROM ENTRYPINT
WHERE ENTRYPINT.entrypointset IN
(SELECT ENTRYPINTSET.entrypointset FROM ENTRYPINTSET
WHERE ENTRYPINTSET.exitpoint IN
(SELECT ENTRYPINT.exitpoint FROM ENTRYPINT
WHERE ENTRYPINT.x_string LIKE "'1997'"))))

OR PAGE.page IN
(SELECT PAGE.page FROM PAGE
WHERE PAGE.chunk IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.cspec IN
(SELECT CSPEC.cspec FROM CSPEC
WHERE CSPEC.c-element IN
(SELECT C-ELEMENT.c-element FROM C-ELEMENT
WHERE C-ELEMENT.c_string LIKE "'1997'"))))

OR PAGE.page IN
(SELECT PAGE.page FROM PAGE
WHERE PAGE.chunk IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.rspec IN
(SELECT RSPEC.rspec FROM RSPEC
WHERE RSPEC.r-element IN
(SELECT R-ELEMENT.r-element FROM R-ELEMENT
WHERE R-ELEMENT.r_string LIKE "'1997'"))))

OR PAGE.page IN
(SELECT PAGE.page FROM PAGE
WHERE PAGE.chunk IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.exitpointset IN
(SELECT EXITPOINTSET.entrypointset FROM EXITPOINTSET
WHERE EXITPOINTSET.entrypoint IN
(SELECT EXITPOINT.exitpoint FROM EXITPOINT
WHERE EXITPOINT.x_string LIKE "'1997'"))))

DELETE * FROM PAGE
WHERE PAGE.page IN SELECTEDPAGES
AND (PAGE.shift = 2)

Figure 7.11: SQL Representation of Example 1
INSERT PAGE.page, PAGE.chunk, PAGE.shift INTO SELECTEDPAGES
WHERE PAGE.page IN (SELECT PAGE.page FROM PAGE)
AND (PAGE.page IN % select page if
(SELECT PAGE.page FROM PAGE) % true vacuously true

DELETE * FROM PAGE
WHERE SELECTEDPAGES.page IN % retain if
(SELECT PAGE.page FROM PAGE % second chunk contains 1998
WHERE PAGE.chunk NOT IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.entrypointset IN
(SELECT ENTRYPOINTSET.entrypointset FROM ENTRYPOINTSET
WHERE ENTRYPOINTSET.entrypoint IN
(SELECT ENTRYPOINT.entrypoint FROM ENTRYPOINT
WHERE ENTRYPOINT.e_string LIKE ‘1998’)))
AND (PAGE.shift = 2)
OR PAGE.page IN
(SELECT PAGE.page FROM PAGE
WHERE PAGE.chunk NOT IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.cspec IN
(SELECT CSPEC.cspec FROM CSPEC
WHERE CSPEC.c-element IN
(SELECT C-ELEMENT.c-element FROM C-ELEMENT
WHERE C-ELEMENT.c_string LIKE ‘1998’)))
AND (PAGE.shift = 2)
OR PAGE.page IN
(SELECT PAGE.page FROM PAGE
WHERE PAGE.chunk NOT IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.rspec IN
(SELECT RSPEC.rspec FROM RSPEC
WHERE RSPEC.r-element IN
(SELECT R-ELEMENT.r-element FROM R-ELEMENT
WHERE R-ELEMENT.r_string LIKE ‘1998’)))
AND (PAGE.shift = 2)
OR PAGE.page IN
(SELECT PAGE.page FROM PAGE
WHERE PAGE.chunk NOT IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.exitpointset IN
(SELECT EXITPOINTSET.entrypointset FROM EXITPOINTSET
WHERE EXITPOINTSET.exitpoint IN
(SELECT EXITPOINT.exitpoint FROM EXITPOINT
WHERE EXITPOINT.x_string LIKE ‘1998’)))
AND (PAGE.shift = 2)

Figure 7.12: SQL Representation of Example 2
aphlbs_read(What, Mode, Answer) :-
    read_tokens(Tokens, _Vars),
    (Tokens = [] -> Answer = end_of_file
     ; (( Phrase =.. [What, A, Tokens, []], call(Phrase))
         -> Answer = A,
             nl, write(What),
             write(' parsing succeeded'), nl,
             aphlbs_feedback(Answer, Mode, _Reply),
             aphlbs_semantics(Answer, SQL)
         ; nl, write(What),
             write(' parsing failed with this token sequence:'),
             nl, nl, write(Tokens), nl, fail
     )
     ).

Figure 7.13: The Back-End Component of PAS

7.4 The Front-End Component of PAS

The front-end component of PAS is responsible for the execution of SQL statement representations of personalisation requests and the dynamic generation of personalised hyperpages. The front-end component is comprised of: a personalisation program; a hyperpage generator; and a program to rewrite terminal strings in rendering expressions.

All front-end processes have been implemented using an application development system for the WWW called Cold Fusion (CF) [Danesh and Kristin, 1999, Forta, 1999]. The principal tools used to develop CF applications are an Application Server and a server-side mark-up language.

The CF Application Server provides the functionality to integrate WWW browser, server and database technologies into WWW applications. The Cold Fusion mark-up language (CFML) is a fourth generation language (4GL) for scripting WWW applications. Based on mark-up tags, CFML provides a server-side programming environment that integrates with HTML [Raggett et al., 1999].

A CF application is a collection of hyperpage specifications (programs) that are authored in CFML and HTML. CFML provides the functionality to control the behaviour of applications, integrate WWW server technologies and dynamically generate the content and presentation of WWW pages before they are returned to a WWW browser.

Broadly, when a hyperpage in a CF application is requested by a WWW browser, it is pre-processed by the CF Application Server. Based on the CFML in the hyperpage, the Application Server executes the application logic, interacts with other server technologies
and then dynamically generates an HTML rendering expression that is returned to the WWW browser.

Figure 7.14 depicts the front-end component of PAS as a collection of CF programs that interact with a CF Application Server. Appendix E contains the complete code for the CF programs that comprise the front-end component of PAS.

7.4.1 Personalisation Program

The personalisation program has been implemented as a CF program. This program retrieves SQL statements stored in a text file and executes these over the relations for hyperpages and hyperpage annotations. Execution is realised via the CF Application Server which provides the functionality to integrate with: local and remote databases; file systems; and files held in directories.

The signature of the personalisation program is

\[
\text{personalisation program} : \text{SQL statement} \rightarrow \text{Outcome}
\]

Appendix E contains the CF program code for the personalisation program.

7.4.2 Generate Hyperpage Program

The generate hyperpage program has been implemented as a CF program. On receiving a request for a hyperpage, this program dynamically generates a hyperpage (rendering expression) by executing a sequence of SQL statements. These statements are used to retrieve sequentially the component parts of a hyperpage (i.e., entry points, C-specs, R-specs and exit points) from a relational database comprised of relations over the relational schemas for hyperpages and hyperpages’ annotations, as described in section 5.5.1. The generate hyperpage program also contains specifications of how content retrieved should be presented.

If a hyperpage (or one of its component parts) is associated with an annotation, then an annotation link is generated to indicate that a particular component part has a note associated with it. To present annotations to the user, a program, generate annotation, has been developed. This program retrieves for a given component part of a hyperpage its corresponding notes.

Information requests (i.e., a request to view an annotation or a subset of the notes that

---

2 An annotation link is a special class of link represented by the image of a pin. When a mouse is moved over the pin, the link is activated and a CF program is run, the purpose of which is to display a subset of notes for a given hyperpage component part.
Figure 7.14: The Front-End Component of PAS

WWW page

WWW Browser

Hyppage request

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW page

WWW p
comprise it) are issued by the user passing a mouse pointer over a link. When such interaction occurs, the generate annotation program displays for a particular hyperpage component part any notes that have been associated.

The signature of the `generate_hyperpage_program` is

\[
generate\_hyperpage\_program : request \rightarrow \text{renderingexpression}
\]

Appendix E contains the CF program code for the `generate_hyperpage_program`.

### 7.4.3 Apply Preferences

The `apply preferences` program has been implemented as a CF program. Given a hyperpage in the form of a rendering expression, apply preferences retrieves for that hyperpage its annotation. If the annotation retrieved specifies one or more post-composition rewrite notes (i.e., has one or more notes, each of which specifies that some terminal symbol be replaced by another terminal symbol), then the apply preference program executes a series of CFML statements that effect a regular expression search and replace over the rendering expression. These statements use as parameters the `from` and `to` regular expressions specified by each post-composition rewrite note. When all specified post-composition rewrites have been executed, the rewritten rendering expression is returned to the UIS that issued the original request for the hyperpage.

This section has outlined the programs developed to implement the front-end component of the PAS. In the following section, the development platform and tools used to implement the PAS are briefly described.

### 7.4.4 Development Tools

The PAS has been developed using the following software tools and system applications:

1. **Microsoft Internet Information Server**: WWW server [Tulloch, 1998];
2. **Cold Fusion Application Development System**: Application development system for the WWW [Danesh and Kristin, 1999];
3. **Microsoft Access**: Database management system [Grauer, 1997];
4. **Netscape**: WWW Browser [Beatty, 1997];
5. **SWI-Prolog**: Logic programming language [Clocksin and Mellish, 1994];
6. **HTML 4**: Hypertext Mark-up Language [Raggett et al., 1999];
7. *Javascript*: Object-based scripting language for developing Internet applications [Danesh, 1997].

In addition to the software development tools above, the Microsoft Windows operating system and various graphics and text-based editors were used in the development of PAS.

### 7.5 Future Work on PAS

The following section briefly outlines future work to be conducted on PAS.

The PAS is at present only a “proof of concept” system and is intended as a test-bed and demonstrator for the results reported in this dissertation. However, the following enhancements will need to be implemented before it can be used as a fully functioning PA-HLBSs.

1. At present the PAS reads in a personalisation request from a text file. These requests are parsed and if well formed, represented as a sequence of SQL statements. Although this approach may be deemed acceptable for the purposes of verifying personalisation requests, it is not a practical method for users to issue personalisation requests. A user-interface component is currently under development that allows users to issue personalisation requests via the same WWW browser that presents personalised hyperpages. This interface is being developed using Javascript, an object-based scripting language;

2. At present the PAS has been designed to be a single-user system. In order to conduct wide-scale empirical tests, extra functionality must be encoded into the PAS to allow multi-user access. Such functionality will require several new relations to store details that allow the tracking of users on an individual basis;

3. At present the PAS is a personalisable HLBS, it does not possess the functionality required to realise adaptivity. Future work on PAS will be concerned with how to implement the architecture described in chapter 6 so empirical tests of the independent effects of user-initiated and system-initiated P&A actions can be carried out;

4. Finally, on completion of a fully working implementation of PAS, large scale empirical tests on the effects of different P&A actions in different contexts are planned to be carried out using the PAS system. Such tests it is hoped will go some way to providing a prescriptive stance as to which P&A actions are more or less effective in different contexts.
This chapter has described the development of PAS, a personalisable HLBS that embodies the main concepts underlying the abstract model of personalisable, adaptive hyperlink-based interaction described in chapters 4 to 6. Chapter 8 aims to give the reader an understanding of how the model proposed compares to, and may be contrasted with, existing approaches to characterising HLBSs and in particular PA-HLBSs.
Chapter 8
Comparing and Contrasting the Proposed Model with Related Work

To the best of the author’s knowledge, the approach contributed in this dissertation, of defining an abstract model of hyperlink-based interaction (thereby isolating the scope for P&A) and then formally inducing from that model the set of P&A actions needed, is novel. However, this research has drawn upon and is related to much existing HLBSs research.

Chapters 1 to 3 sought to provide the reader with a detailed understanding of existing research into characterising PA-HLBSs. Chapters 4 to 6 detailed a formal model for personalisable hyperlink-based interaction that forms the major contribution of the work reported in this dissertation. This chapter does not repeat the survey of research results reported in chapters 1 to 3, but aims to give the reader an understanding of how the model proposed compares to, and may be contrasted with, existing approaches to characterising HLBSs and in particular PA-HLBSs.

8.1 Introduction

Due to the large body of research into HLBSs, this chapter makes no attempt to be comprehensive in scope, but does identify many representative examples of hyperlink-based models and systems that relate to the work reported in this dissertation to varying degrees. It is hoped that this chapter will further place in context the contributions made by the proposed model of personalisable, adaptive hyperlink-based interaction.

Section 8.2 compares and contrasts the proposed model with that of early HLBSs modelling research. The motivation for this is that several of these early proposals, in particular the HAM and Dexter models, have become reference standards, against which much research has been based and is compared to.

Following this, section 8.3 compares and contrasts OHLBSs research efforts with that of the work reported in this dissertation. This section highlights the fact that although
the work reported here shares many of the underlying principles of OHLBSs modelling research, there are important differences. Furthermore, the central motivation of the research reported here, namely tailoring hyperlink-based interaction, differs radically from that of OHLBSs research which, broadly, can be summarised as research into integrating hyperlink functionality with existing and future computing environments.

Section 8.4 compares and contrasts the model proposed in this dissertation with alternative approaches to modelling and implementing PA-HLBSs. When assessing the ability of the model proposed to reflect recognised methods of personalisation and adaption, it is argued that the formal approach taken here has yielded significant benefits.

The model proposed in chapters 4 to 6 is, in part, based on languages for the specification of hyperpages (see section 4.4), hyperpage annotations and personalisation requests (see section 5.4). This approach is similar in some respects to the ideas behind the development of mark-up languages for the defining and processing of structured documents, such as SGML, XML, DHTML and SMIL. Section 8.5 therefore highlights similarities and differences between several existing document language proposals and the work reported in this dissertation.

The chapter concludes by summarising the similarities and differences with existing HLBSs research.

### 8.2 Comparisons with Early HLBSs Modelling Research

Early research into HLBSs models addressed the question of how to model instantiations of existing HLBSs. Such research sought to model the common characteristics (structural, behavioural and navigational) of such systems. The following section compares and contrasts the model proposed in this dissertation with that of early proposals of models of HLBSs.

#### 8.2.1 Comparisons with the HAM

The HAM [Campbell and Goodman, 1987] was the first attempt to define an abstract model within which HLBSs could be expressed. The model describes a transaction-based storage system. The definition of the HAM consists of a description of HAM objects and the operations that can be applied to them. The HAM describes a lower level machine tied closely to the storage (file) system while having a looser connection to applications and user interfaces (see subsection 3.2.1 for a detailed discussion on HAM). Although the approach taken in devising the HAM is related to the work reported in this dissertation, it differs in several important respects:
• The HAM views a hyper-network as a database of objects and the operations described are primarily those to manipulate this database (e.g., create, delete, get, filter). This approach can be contrasted with the model proposed here, which attempts to identify and characterise those operations which are unique to HLBSs as opposed to those operations which take place between a core of hyperlink functionality and a storage system;

• Although the HAM views a core of hyperlink functionality (the HAM itself) as being loosely tied to applications and user interfaces, it views the host file system’s structure and contents as being tightly coupled to the HAM. This view can be contrasted with the work reported here, in which both user-interface and database functionality are viewed as being loosely coupled to possibly remote servers whose internal structure may be unknown. How such coupling takes place is viewed as a side effect of the query specification languages and mark-up languages used to specify the content and presentation of hyperpages. Furthermore, the definition of content specifications (see section 4.2) allows for the unconstrained use of data query languages. This level of functionality is not modelled by the HAM;

• It can be seen from the specification of the HAM that its authors realised the need for features to allow personalisation of hyper-networks. The filtering mechanism described in the HAM allows subsets of HAM objects to be extracted from hypernetworks (HAM graphs), thereby allowing a user to specify a subset of objects to be presented. However, this approach to personalisation is primarily data-oriented and could be broadly understood as creating alternative views of an underlying database. In contrast, the work reported here models personalisation of a core of hyperlink functionality, as opposed to modelling the rewriting (personalisation) of database queries.

8.2.2 Comparisons with the Dexter Model

The Dexter model [Halasz and Schwartz, 1990, Halasz and Schwartz, 1994] has become the foundation for much research into HLBSs modelling and is a reference standard against which many HLBSs are compared. The model is divided into three layers; the storage layer, the within component layer and the run-time layer. The main focus of the model is the storage layer which describes a database that is composed of a hierarchy of data containing “components” called nodes interconnected by links. The storage layer therefore models the basic node/link network structure of a hyper-network.

The Dexter model is similar to the work reported in this dissertation in that it is an attempt to explore the concepts and operations which take place within HLBSs. For example, the Dexter model describes an interface mechanism for addressing (referring to) locations or items within the content of an individual component. This mechanism, known as anchoring, provides a clean separation between the storage and within-component layers.
Anchoring provides the functionality to allow links to be made between documents and also parts of documents.

In the model proposed in this dissertation a content specification, C-spec, plays a comparable role to the Dexter model’s anchoring mechanism. A C-spec defines content which is to appear in a hyperpage. A C-spec takes the form of data values or requests to DBSs (i.e., query expressions which DBSs can evaluate into data values which are served back). A C-spec may be as simple as a number or a string and as complex as a sequence of complex queries which are to be sent to a variety of DBSs, possibly in many different query languages and using many different client-server protocols.

In the Dexter model, presentation specifications provide an interface between the storage layer and the run-time layer (i.e., UIS). Such specifications are the mechanism by which information about components presented to the user can be encoded into the hyper-network (at the storage layer).

In the model proposed, rendering specifications, R-specs, define how content is to be rendered by a UIS. An R-spec takes the form of formal text (i.e., mark-ups) in a language which the intended UIS can render. This renderable text may be interspersed with template variables that act as place holders for content as defined by the C-spec with which the R-spec is paired. An R-spec can therefore be used to specify how content is to be presented in a manner similar to that of presentation specifications in the Dexter model.

A further similarity between the Dexter model and the model proposed here is that both are formally described. The formalisation of the Dexter model has helped it to be widely understood and provable. This was a motivating factor for formalising the model proposed in this dissertation.

Although both the Dexter model and the model presented here have several shared aims, its focus differs from the work reported here in several important respects:

1. The Dexter model does not address the issue of tailoring the interaction process that takes place between users of HLBSs and the system itself. No aspects of personalising the interaction process are explicitly modelled and no consideration is given to users of HLBSs that may have differing information retrieval needs, user goals and backgrounds;

2. In the Dexter model the storage layer models a hyper-network as a known database of nodes and links. As such, the amount, scope and form of the hyper-network can be said to be deterministic. In contrast, the work reported here views a hyper-network as a collection of specifications which determine how to retrieve content, which data source should be used and how retrieved content should be presented. As such, the hyper-network’s content, scope and form is undeterministic;

3. The Dexter model sees HLBSs as being layered and including inter-connected user-
interface and data storage layers. An implicit assumption is made that these layers are aware of and are tightly bound to each other. In contrast, the proposed model does not view the functionality provided by these layers as being tightly bound. Instead this functionality is provided by a loosely-coupled core of hyperlink functionality served by a variety of possibly remote user-interface and content servers. As such the proposed architecture reflects more accurately those of many OHLBSs currently being deployed (i.e., WWW). A comparison of the Dexter model’s architecture with that of the model proposed in this dissertation is shown in Figure 8.1;

Figure 8.1: A Comparison of the Dexter Model and the Abstract Model Proposed

4. The Dexter model implicitly assumes that applications making up the within-component and run-time layers are known. This monolithic view cannot be assumed to be true. The array of different presentation, link anchoring and data storage mechanisms that could potentially be employed by OHLBSs implies that their behaviour and computation should not be modelled as an integral part of a HLBS. The architecture for HLBSs proposed in this dissertation may be extended to support such an array of presentation, link anchoring and data storage mechanisms;

5. The Dexter model does not make a clear distinction between mechanisms used to reference composite components (hyperpages) and those mechanisms used to identify the atomic components which they comprise. The model proposed in this dissertation provides a detailed formalism that separates the referencing of all component parts of a hyperpage (i.e., chunks, entry-points, C-specs, R-specs and exit-points);

6. The within-component and run-time layer functionality specified by the Dexter model lies beyond the scope of the model defined within this dissertation.

In summary, although the Dexter model has provided a solid foundation for research into closed HLBSs, it is less appropriate for representing OHLBSs. Furthermore, it does not explicitly address issues of personalising the user interaction.
8.2.3 Comparisons with other Early Models of HLBSs

The Trellis Hypertext Reference Model [Furuta and Stotts, 1990] is an attempt to define HLBSs in terms of different levels of abstraction. Each level describes the representation of part or all of a hyper-network.

The Trellis model has several similarities to the model proposed in this dissertation. In a manner comparable to the model proposed, the Trellis model allows the presence of links to be made conditional on user actions. Such functionality enables controls to be placed on the accessibility of hyperdocuments. The approach of deliberately excluding the content and user-interface aspects of HLBSs is similar to that of the model proposed. However, in contrast the model proposed encompasses the dynamic aspects of HLBSs and views such aspects as central to what differentiates HLBSs from other kinds of information retrieval systems.

The Formal model proposed in [Lange, 1990] is a data model with many similarities to the Dexter model. A fundamental difference between this model and the model proposed in this dissertation is that the Formal model places a strong emphasis on textual information, making it inappropriate to describe HLBSs in general. The model proposed in this work attempts to provide an abstract definition of hyperlinked items, thereby making it more widely applicable.

8.3 Comparisons with OHLBSs Modelling Research

The main goal of research into OHLBSs has been to push forward the architectural possibilities of HLBSs by both reacting to and showing the demand for more advanced technology platforms. This research has in the main been directed towards the design, modelling and development of OHLBSs. Figure 8.2 lists representative examples of such research.

<table>
<thead>
<tr>
<th>Model</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMS</td>
<td>[Akscyn et al., 1988]</td>
</tr>
<tr>
<td>Sun's link service</td>
<td>[Pearl, 1989]</td>
</tr>
<tr>
<td>PROXHY</td>
<td>[Kacmar and Leggett, 1991]</td>
</tr>
<tr>
<td>Microcosm</td>
<td>[Davis et al., 1992a]</td>
</tr>
<tr>
<td>Multicard</td>
<td>[Rizk and Sauter, 1992]</td>
</tr>
<tr>
<td>HDM</td>
<td>[Garzotto et al., 1993]</td>
</tr>
<tr>
<td>SP3</td>
<td>[Leggett and Schnase, 1994]</td>
</tr>
<tr>
<td>Devise Hypermedia Framework</td>
<td>[Grønbæk and Trigg, 1994a]</td>
</tr>
<tr>
<td>HyTime</td>
<td>[De Rose and Durand, 1994]</td>
</tr>
<tr>
<td>RMM</td>
<td>[Isakowitz et al., 1995]</td>
</tr>
<tr>
<td>Chimera</td>
<td>[Anderson et al., 1994]</td>
</tr>
<tr>
<td>HyperDisco</td>
<td>[Wiil and Leggett, 1996]</td>
</tr>
</tbody>
</table>

Figure 8.2: Representative Examples of OHLBSs Modelling Research

The wealth of research in this area makes an exhaustive examination virtually impossible.
However, in this section, several of the most salient aspects of the research related to that which is reported in this dissertation are discussed.

Although much work has been conducted into characterising the functionality which OHLBSs could provide to hyperlink unaware applications (i.e., applications that have no notion of linking), in contrast to various OHLBS proposals, the work reported here views such a characterisation, at a suitable level of abstraction, to be an essential starting point from which to induce a set of comprehensive P&A actions.

The approach of OHLBSs researchers to characterise a HLBS as being comprised of a core of functionality that is integrated with various external servers (e.g., data and link servers) [Wiil and Østerbye, 1994, Davis and Lewis, 1996, Nürnberg and Leggett, 1997] is shared by the work reported in this dissertation. Whereas in the model proposed, it is assumed that a HLBS is a client of a user-interface and database servers, nothing in the model comes into conflict with a move towards making it a client also of link servers. Furthermore, the dynamic fetching of content which may be denoted by a hyperpage specification exemplifies how the fetching of navigation possibilities might also be incorporated into the proposed model.

When examining specific proposals for OHLBSs, the work on extending the Dexter model to include freely defined embedded references, reported in [Davis, 1995, Brown and Brown, 1995, Grønbæk and Trigg, 1996], is similar to that of the work reported here. In [Grønbæk and Trigg, 1996], the definition given of a Locationspecifiers that allow the specifying of a location of a component across HLBSs is comparable to the definition given for an exit point in section 4.2. The main difference between the two is that in the work reported here, references to hyperpages and their component parts are defined at a level of abstraction that allows for their authoring in any formal text. No restrictions are placed on the form of such references with such decisions being the responsibility of the designer of the hyperpage. Similarly, the definition of Referencespecifiers given in [Grønbæk and Trigg, 1996] is comparable with the definition for entry points given in section 4.2.

The ISO standard HyTime (Hypermedia/Time-based Structuring Language) specifies the representation of hyperdocuments in a presentation independent format. HyTime allows for the representation and synchronisation of static and time-based hyperlinked information [Newcomb et al., 1991, Newcomb and Newcomb, 1992, De Rose and Durand, 1994]. HyTime is defined as a subset of the Standard Generalised Mark-up Language (SGML) which defines the structure of electronic documents in general [Heimburger, 1994]. HyTime supports addressing by name, by position in the document and by semantic construct. Links can be established to documents that conform to HyTime as well as those that do not. HyTime also provides a document query language.

HyTime allows all kinds of multimedia and hyperlink technologies (whether proprietary
or not) to be combined in a single application. It addresses only the issue of information interchange and not the standardisation of presentation, user-interfaces and query languages. Objects in a HyTime hyperdocument can include formatted and unformatted documents, audio and video segments, still images, animations and graphics.

Although the aims of HyTime and the work reported in this dissertation differ, several similarities can be drawn. HyTime and the work reported here model hyperpages as specifications in which external content may be referenced. The level of “openness” that HyTime allows, in terms of the kinds of media and technologies that can be referenced by hyperpage specifications, is comparable to that which is allowed by the model proposed.

In both models, addressing of hyperpages and their component parts can be specified using semantic constructs. Furthermore, the ability to incorporate links which are specified using an external formal text is also shared.

However, HyTime is a mature model which has resulted in the development of various HLBSs (e.g., [Koegel et al., 1993, Carr et al., 1994, Buford, 1996]). HyTime was originally devised to address problems of data interchange between HLBSs. As such, HyTime provides many facilities not incorporated in the model proposed in this dissertation. In particular, the specification for positioning and projecting objects in time and space, detailed by HyTime, is not addressed in this work.

Furthermore, HyTime is a meta-language in that it may be used to code not only individual hyperdocuments, but also document sets to which they belong. A HyTime document set is defined by an SGML document type definition (DTD) which defines a specific syntax in terms of SGML constructs that documents must follow. HyTime therefore inherits from SGML the use of DTDs. In contrast, the model proposed here specifies a tentative characterisation of hyperpage specifications, the primary purpose of which is to form the basis for the formal definition of personalisation requests over these specifications. It is reasonable to postulate that a meta-language such as SGML or HyTime could be used to implement the hyperpage specifications defined by the proposed model.

The DHM is an open hyperbase system [Grønbæk and Trigg, 1994a, Grønbæk et al., 1994] that enables the tailoring of its data model. The source code of DHM’s data model manager and session can be tailored at run-time. Such tailoring is achieved using an interpreter and an object-oriented framework that represents the system’s hyper-network. Content is stored in an object-oriented fashion that may be internal or external to the system. Furthermore, it is assumed that viewers are available to edit this content.

In the DHM system, integration with third party applications is realised by the use of wrappers specified by viewers. The form of anchors used to specify the inter-connection of hyperlinked items is left undefined. It is proposed that the realisation of such anchors is dependent upon the viewer that renders them. The DHM system supports a resolver function similar to that of Microcosm [Davis et al., 1994] for computing the end points of
links. This function is flexible enough to be tailored so that different link end points are made available in different contexts.

The approach taken of devolving responsibility for the form of anchors to particular view- ers clearly parallels the approach taken to specifying entry points and exit points (see section 4.2 for further details). However, tailorability with the DHM system is at the source code level and as such requires the user to have knowledge of the internal structure of the DHM system and in particular, the object-oriented framework upon which it is based. In contrast, in the work reported here it is the collection of hyperpage specifications and annotations that are tailored and the user is not required to have any knowledge of the internal operations which act upon them. All that is required is for users to be able to issue well formed personalisation requests.

8.3.1 Representing PAS using the Flag Taxonomy

The Flag taxonomy model [Osterbye and Wiil, 1996] provides a framework to classify and concisely describe individual HLBSs in a system independent way. The aim of the Flag taxonomy is to clearly distinguish between the storage, run-time, structure and contents aspects of OHLBSs. The model is comprised of four functional modules and four protocols. Each functional module provides functionality to be used by its neighbouring functional modules through the available protocols. Figure 8.3 depicts the proposed model represented using the Flag taxonomy.

Figure 8.3: The Proposed Model Represented using The Flag Taxonomy

Figure 8.3 shows that in the model proposed no protocols are specified for the linking, storage and presentation of hyperlinked information. A further conclusion that can be drawn is that although the Flag taxonomy is suitable for distinguishing between the storage, run-time, structure and contents aspects of OHLBSs, it is unsuitable for distinguishing a core of hyperlink functionality from separate layers of functionality that model personalisation and adaptivity.
8.4 Comparisons with PA-HLBSs Modelling Research

The work reported in this dissertation is most closely related, in the sense of having shared goals, to research into PA-HLBSs. The aim of such research is to increase the functionality of HLBSs by making the user interaction process personalisable. The following section compares and contrasts the proposed model with that of other approaches to modelling and implementing PA-HLBSs.

Attention should be drawn to the fact that to the best of the author’s knowledge, no other researchers have taken the view or formally proved that every design decision made when specifying a hyperpage can be overridden and, concomitantly, characterises how this process of overriding can take place by defining a language for personalisation actions and describing how it effects ownership transfer.

8.4.1 Comparisons with the AHAM

The Dexter-based Reference Model (AHAM) proposed in [de Bra et al., 1999] aims to encompass many of the adaptive features supported by adaptive HLBSs. The model presents a set of formalisms to define several adaptive techniques. In AHAM adaptation is based on a domain model, a user model and a teaching model which consists of pedagogical rules. A simple formalism is provided which illustrates how designers can write pedagogical rules about concepts in such a way that they may be applied automatically when these concepts are retrieved for presentation.

Although several of the aims of AHAM and those of the model proposed in this dissertation are shared and at present there is little published work on the AHAM, it appears that there are fundamental differences not only between each model’s architectures, but also the underlying theoretical basis upon which they have been designed.

The theoretical basis of AHAM seems to imply that issues of personalisation and adaptivity can be compounded together and as such the model does not make any distinction between the two. In contrast, the work reported here proposes an alternative characterisation in which the issues and effects of personalisation can be modelled and studied independently from those concerned with adaptivity.

In the AHAM, notions of personalisation and adaptivity are not formally characterised. It is assumed that what is meant by personalisation and adaptivity in the context of HLBSs is agreed and known. This view is not shared by the author of this dissertation.

In the work proposed here, it is argued that in many implementations of PA-HLBSs it is still unclear which aspects are possible to personalise and what the ramifications of P&A actions issued are. Furthermore, the level and types of functionality offered by the adaptive functions of such systems varies widely. The lack of an accepted terminology, taxonomy
or underlying principles for personalisation has meant that different implementations have been achieved using radically different approaches and have implemented vastly different sets of P&A actions.

This dissertation argues that the scope for P&A actions in HLBSs has yet to be formally defined. Furthermore, it argues that this scope comprises exactly the emergent properties of HLBSs when these are viewed as only loosely coupled to a variety of servers. It is the specificity of HLBSs, and hence of the scope for P&A actions they might support, that the research reported in this dissertation seeks to identify at an abstract level above that of concrete systems. In the work reported here, notions of personalisation and adaptivity are meticulously defined and, in the case of personalisation, formalised. The set of P&A actions offered is induced from a formal characterisation of a formalised core of hyperlink functionality that represents the emergent properties of HLBSs.

The focus of the AHAM is on representing techniques that have been implemented, several examples of which are given. In contrast, the work reported here aims to induce a set of P&A actions from a formal definition of a core of hyperlink functionality. The approach taken, therefore, not only shows how implemented techniques can be modelled, but also provides a principled and exhaustive elicitation of the space of possible P&A actions. Broadly, the approach taken in the proposed model has yielded one complete picture of what should be the subject of personalisation, what P&A actions could be made possible and what is the formal consequence of issuing these P&A actions. The AHAM has not.

The AHAM views the adaptive function as an integral part of the model but provides only very general statements on its form or operation. Furthermore, no semantics are detailed for the operation of the adaptive function described by the AHAM. In contrast, the model proposed here views the adaptive function as a “black box” whose purpose is to monitor events, make conjectures in the light of a user’s goals and history and then issue personalisation requests based on the result of these conjectures. However, to illustrate how adaptivity may be realised, chapter 6 details an architecture, a procedural design and a formalism for an adaptive function based upon active database theory.

The AHAM presents a small set of formalisms based upon a set of pedagogical rules. The pedagogical rules modelled by AHAM are similar in some respects to those ECA rules modelled by the work reported here. However, the ECA rules are defined as an active rule set over an active database representation of a hyper-library (see section 6.4). Furthermore, in the work reported here a detailed formalism is given of not only active rules, but also hyperpages, hyperpage annotations, personalisation requests, user goals and histories.

In summary, the AHAM aims to illustrate how adaptive techniques may be implemented through the use of a set of simple formalisms of these techniques. However, in the model no distinction is made between the processes of P&A and no semantics are presented to
formally define their meaning. Furthermore, the adaptive engine which is said to drive the adaptive process is only briefly described in terms of the functionality it should possess. No detailed formalisms are given of its operation or the semantics of personalisation requests.

8.4.2 Comparisons with Implementation-led Approaches to PA-HLBSs

The implementation of PA-HLBSs is not new (see figure 8.4 for a representative sample), however, little research has been conducted into formalising personalisation within such systems. However, much previous work has concentrated on providing personalisation and adaptivity features via data querying mechanisms, such as in [Afrati and Koutras, 1990, Hockemeyer et al., 1998]. Although this work shows how personalisation may be achieved, it can be argued that in these works personalisation is a side-effect of DBMS functionality rather than hyperlink-based interaction.

<table>
<thead>
<tr>
<th>2L670</th>
<th>[Bra and Calvi, 1998]</th>
</tr>
</thead>
<tbody>
<tr>
<td>RATH</td>
<td>[Hockemeyer et al., 1998]</td>
</tr>
<tr>
<td>PEBBA</td>
<td>[Milosavljevic and Oberlander, 1998b]</td>
</tr>
<tr>
<td>C-Book</td>
<td>[Kay and Kummerfeld, 1995]</td>
</tr>
<tr>
<td>KN-AHS</td>
<td>[Kobsa et al., 1994]</td>
</tr>
<tr>
<td>MetaDoc</td>
<td>[Boyle and Encarnacion, 1994]</td>
</tr>
<tr>
<td>PUSH</td>
<td>[Höök et al., 1996]</td>
</tr>
<tr>
<td>Hynecosum</td>
<td>[Vassileva, 1996]</td>
</tr>
<tr>
<td>ELM-ART</td>
<td>[Brusilovsky et al., 1996a]</td>
</tr>
<tr>
<td>Adaptive HyperMan</td>
<td>[Mathe and Chen, 1996]</td>
</tr>
</tbody>
</table>

Figure 8.4: Representative Examples of PA-HLBSs

However, in [Beeri and Kornatzky, 1990], an approach comparable to that reported in this dissertation is presented for personalising HLBSs using a logical query language. This language permits the formulation of structural queries which enable parts of a hyper-network to be retrieved based on a specification of their structure. The language is propositional and includes a general notion of a qualifier of the form appropriate for hypertext networks. Qualifiers are used for expressing formulas of the form “for most paths from the current node claim x holds”. Formulas in the language are used for a declarative definition of user-tailored views of a hypertext system.

The approach taken in this dissertation differs from that in [Beeri and Kornatzky, 1990] in several respects. The model presented in [Beeri and Kornatzky, 1990] is based on a labelled directed graph. The labelling of nodes and links is used to represent the information they contain using two primitives: attributes, which define the type of information (e.g. text, graphics); propositions, which describe the contents and semantic role (issue, discussion) of nodes and links in particular applications. In this approach, descriptions of the hyper-network are combined with the network itself. In contrast, in the approach reported here, the two are separate and may be manipulated independently. Furthermore, the logical query language in [Beeri and Kornatzky, 1990] focuses on the querying of data sources,
whereas, in this dissertation a formal language is proposed for manipulating hyperpages
and hyperpage annotations, not for querying data sources.

In [Hockemeyer et al., 1998, Hockemeyer and Albert, 1999], a relational WWW adaptive
tutoring hypertext environment (RATH) is proposed. RATH is an adaptive tutorial sys-

Like one page of a document, here is a text description of its content:

whereas, in this dissertation a formal language is proposed for manipulating hyperpages
and hyperpage annotations, not for querying data sources.

In [Hockemeyer et al., 1998, Hockemeyer and Albert, 1999], a relational WWW adaptive
tutoring hypertext environment (RATH) is proposed. RATH is an adaptive tutorial sys-
tem that combines a relational model for the structure of a hyper-network, based on the
Dexter model, with the theory of knowledge spaces from mathematical psychology. Using
prerequisite relationships between different items in a domain of knowledge and informa-
tion about users’ current knowledge state, RATH adapts the presentation of links in a
hyperdocument on an individualised basis. Paths through the hyper-network (i.e., a sub
hyper-network) may also be stored and these activities are implemented in an off-line
manner.

The operation of the RATH system is implemented using the mark-up language HTML on
the WWW. Users communicate via a WWW browser with a WWW server. This server
uses Common Gateway Interface (CGI)¹ scripts to build user adapted components by
modifying (using relational database functionality) components from the hyper-network
according to internally stored user data. The user adapted component is then transferred
via the WWW server to the user’s browser and presented to the user.

The architecture of the RATH is similar in several respects to that of the PAS system
described in chapter 7. The use of the relational data model to represent hyperpages is
shared. However, the RATH system implements hyperpages using the Dexter model. In
contrast, the work reported here models hyperpage and hyperpage annotations as specifi-
cations using the relational model (see section 5.5.1).

The use of a WWW browser in the role of a UIS and a WWW server as an interface
between a UIS and a database is implemented in a manner that can be compared to
the PAS system. The role of CGI scripts to dynamically generate hyperpages on an
individualised basis is also similar to that played by the CFML mark-up language in PAS.

Significant differences between the two systems include the following:

- The back-end of PAS is responsible for parsing and then interpreting the meaning
  of a personalisation request as a sequence of SQL expressions. The RATH system
does not implement such functionality;

- In the PAS system personalisation is realised as a result of evaluating not only a
  hyperpage’s content, but also annotations associated with it. These annotations
may specify not only additional information on a component part of a hyperpage
but also the rewriting of terminal strings found within that page (i.e., rewriting GB
to UK). This level of personalisation is not implemented in RATH;

¹CGI scripts are programs which allow the fetching of content from local and remote data stores and
the transfer of that content back via WWW servers to WWW browsers [Kim, 1996].
• The RATH system does not implement user-initiated tailoring. All personalisation
   is system-initiated and as such there is no opportunity for a user to interfere in the
   personalisation process;

• It is unclear from the RATH proposal the range of possible P&A actions that are
   possible. P&A actions are implemented as a CGI program and few details are
   provided of its internal structure or operations.

In [Loke and Davison, 1996], LogicWeb, a system that integrates structured logic pro-
gramming and the WWW, is described. LogicWeb enables programmable behaviour and
state to be incorporated into hyperpages, allowing them to be viewed as modules or objects
with state.

LogicWeb renders a hyperpage as a “live” information unit, able to determine its own
response to user queries and modify its availability of links. The amalgamation of logic
programming and the WWW makes possible the dynamic generation of adaptable hy-
perpages. Hyperpages are implemented as logic programs, termed LogicWeb programs.
Special operators are available in LogicWeb for the retrieval of LogicWeb programs and
to invoke goals within them. Implementing hyperpages as LogicWeb programs also en-
ables their contents to be accessed as logic programming facts. For instance, information
about the links in a page can be represented as a set of facts, so providing an additional
layer of abstraction beyond the text of a hyperpage. A prototype system is described
in [Loke and Davison, 1996], which extends WWW browser technologies with LogicWeb
capabilities via a Common Client Interface.

The use of LogicWeb to combine the logic programming and the WWW technologies is
comparable to the use of prolog in PAS. However, LogicWeb is primarily used to apply
rules to LogicWeb programs (i.e., hyperpage and link structures). In contrast, Prolog
is primarily used to implement a parser for hyperpages, annotations and personalisation
requests. The Prolog implementation in PAS is not at present tightly integrated with
the WWW component of PAS, although, in latter implementations a tighter coupling is
envisaged.

Natural Language Generation (NLG) systems can be characterised as consisting of two
components, a text planner and a surface realisation component. The text planner is
responsible for the preparation of a discourse plan using facts from a knowledge base.
This involves making decisions based on a set of discourse goals. The text planner must
decide the relevance of content in a particular situation and then compose this content
as a coherent text. The discourse plan is realised as natural language utterances by a
surface realisation component. This makes use of knowledge of the language’s grammar
and lexicon to produce well-formed utterances conveying the required content.

Several HLBSs have been developed using NLG techniques (e.g., ILEX and PEBA
[Milosavljevic et al., 1996, Milosavljevic and Oberlander, 1998b]) to tailor hyperlink-based
interaction by giving users the freedom to perform high-level discourse planning. A key characteristic of these systems is that the content of the hyper-network is dynamically created at run-time in response to user requests. The knowledge base of such systems is comprised of information about concepts in the hyper-network. The NLG component of such systems is then responsible for selecting which elements of the knowledge base are important for creating the required hyperdocument.

The surface realisation component of such systems is often also responsible for the presentation of links that represent follow-up questions which the user can ask. Each link indicates a new discourse goal to the system.

In [White, 1998] Exemplars, a rule-based object-oriented framework with capabilities to tailor dynamically generated hyperpages is presented.

The approach taken in [White, 1998] is closely related to that of PA-HLBSs that use NLG techniques to tailor hyperlink-based interaction. The Exemplars framework views the dynamic generation of personalised hyperlink-based interaction as essentially an expert-systems task, in which specialisation plays a key role. The framework enables the designer to determine the behaviour of a HLBS by writing a set of object-oriented text planning rules and arranging them into a specialisation hierarchy, where more specialised rules can augment or override more general ones.

The text planning rules, called exemplars, aim to capture an exemplary way of achieving a communicative goal in a given communicative context. Each rule has a condition and an action. The condition defines the applicability of the rule in terms of tests on the input data, the discourse context and a user model, whereas the action defines what content to add to the current output and how to update the discourse context and user model.

When assessing the ability of the model proposed in this dissertation to reflect recognised methods of personalisation, it can be argued that the formal approach taken has yielded significant benefits. In the model proposed, all hyperpage design decisions may be the subject of personalisation requests. Furthermore, results of this approach seem to allow the vast array of adaptive techniques detailed in [Brusilovsky, 1996b] to be clearly and simply modelled.

The notion of hyperpage annotations presented in section 5.4 can be contrasted with that of [Egov et al., 1994] in which the ELBI library system is discussed. In particular, the formation of personal libraries, reflecting their owner’s personal taste and the ability to mark out arbitrary, semantically important fragments (nodes), seems a practical application of the approach to personalisation proposed in this dissertation.

In [Egov et al., 1994], users are able to annotate any given piece of information and to store such annotations as personal hypersummaries. Hypersummaries are characterised as the result of a user’s efforts in eliciting the information he/she needs and organising it
into appropriate structures. This clearly has parallels with the notion of a hyper-library of annotations (which are comprised of notes) which, in part, reflect information needs. Furthermore, the notion of storing the total domain (all possible hyperpages) in a hyper-library is also shared with the work reported here.

The technique of link annotation [Brusilovsky et al., 1996a, Calvi and de Bra, 1997] (the representation of availability of links and the incorporation of visual clues, indicating their status or purpose) can be clearly represented in the proposed model. The notion of not only controlling the visualisation of links but also the ability to manipulate them (i.e., allowing them to be visible according to the appropriateness of the situation [Calvi and de Bra, 1997]) can also be represented.

Personalisation techniques, such as the personalisation of content [Beaumont, 1994, Boyle and Encarnacion, 1994], can be clearly represented. Refinements such as selective content (i.e., hiding user parts of information about a particular concept which the user has expressed a wish not to see) and additional explanations as in KN-AHS [Kobsa et al., 1994] and Anatom-Tutor [Beaumont, 1994] can be modelled.

The grammar for personalisation requests may be used to model not only comparative content, but also content variation (i.e., providing a variant of the content deemed appropriate as implementation in [Beaumont, 1994, Kay and Kummerfeld, 1995]).

### 8.5 Comparisons with Generalised Mark-up Languages

Generalised Mark-up Languages may be understood as explicit mapping tools of document structures. As such, they descriptively identify and then map each meaningful document structure by a function (i.e., paragraph, table or graphic). Although primarily used for textual interchange, several proposals have been made which allow for the specifying of optional content and calls to external functions (i.e., XML [Bray et al., 1998] and SMIL [Hoschka, 1998]).

Documents are usually logically structured by sets of markers (tags) which are placed at the beginning and end of each meaningful document structure. These markers denote explicit rules where specific content can or must exist.

An important motivation for these mark-up languages is the need to separate the content of a hyperdocument from its formatting. Such a separation is seen to be beneficial in that it allows relatively easy editing of hyperdocuments and in particular, revisions of either content or formatting do not necessitate changes in the other.

Generalised Mark-up Rule Sets (GMRSs), referred to as meta-languages, are used to write mark-up languages. Document components including structure and element names, called document type definitions (DTD), are declared when writing a mark-up language. The
most popular GMRS is SGML [Heimburger, 1994] which offers a large, complex set of information management features.

8.5.1 Comparisons with XML and SMIL

The work done by W3C, the WWW Consortium, on the extensible mark-up language XML [Bray et al., 1998] bears some resemblance to the work reported here. XML is a simple dialect of SGML. Both approaches seek to divide a hyperdocument into units which either specify data or delineate regions where data may be obtained.

The use in XML of arbitrary-value pairs associated with a document’s structure can be contrasted with the notions of an annotations note as described in section 5.4.

The functionality of the XML processor is not dissimilar from that of the composition function within the H-region. However, there are many more differences than similarities.

In this dissertation an abstract model of personalisable, adaptive hyperlink-based interaction is proposed (i.e., it addresses architectural issues and not just specification ones), while the developers of XML seek to build a concrete scripting language based on, and compatible with, SGML.

In the work reported here, the goal is personalisation of hyperlink-based interaction, whereas the goal of XML is to allow documents to express local semantics, without any provision for the personalisation of hyperdocuments not already owned by the user.

The model proposed here is compatible with XML insofar as no assumptions are made on the semantics of UIS and DBS strings, therefore any facilities provided by XML in that respect can be embraced by the proposed model.

The Synchronised Multimedia Integration Language (SMIL) has been recently proposed by the W3C working group on synchronised multimedia (SYMM) [Hoschka, 1998]. SMIL is a language for describing interactive multimedia, distributed on the WWW. SMIL is an XML-based language and therefore its rule set is drawn from SGML.

SMIL contains two programming constructs, \textit{SWITCH} and \textit{system test attribute}, that can be used to express adaptive content selection and synchronisation relationships among media elements. Using SMIL, a designer can describe the temporal behaviour of multimedia presentations, associate links with media objects and describe the layout of presentations.

The \textit{SWITCH} allows a series of alternatives to be specified for a particular piece of content, one of which is selected by the run-time environment for presentation. The \textit{SWITCH} provides a conventional branching structure that allows alternatives to be specified for a particular piece of content at authoring time and the evaluation of these alternatives so that one is selected at run-time for presentation.
In SMIL, system test attributes provide a mechanism for controlling adaptive behaviour. System test attributes consist of a set of pre-defined (primarily system-related) attributes that describe dynamic aspects of the environment which can then be tested at run-time (i.e., include x if test attribute y is true). These attributes are comparable with the attribute set defined for hyperpage annotations (see section 5.4).

8.6 Summary of Related Work

In summary, the majority of early HLBSs modelling research characterises HLBSs as closed monolithic systems in which a tight coupling of user-interface and database components can be observed. In contrast, the work reported here does not. It views a HLBS as being comprised of a core of hyperlink functionality loosely coupled to a variety of servers. Personalisation of this core is modelled as the rewriting of hyperpage specifications and annotations that, in part, comprise this core.

Although OHLBSs research also views a HLBS as comprising a core of hyperlink functionality, this research is concerned with integrating this core with third party applications by providing facilities to make third party applications hyperlink aware. However, the computing landscape upon which much of this research is based is changing rapidly. The notion of a hyperlink structure can now be found in many applications (i.e., Word, Access, Emacs, Unix and Windows-based help systems) and is incorporated into several operating systems (Windows 98 and Linux). Furthermore, the popularity of the WWW as the primary hyperlink computing environment and protocols such as HTTP and TCP/IP being incorporated into many computing environments and applications suggests that issues related to the specialisation of protocols for interacting HLBSs with third party applications have been addressed. In contrast to various OHLBSs proposals, the work reported here views such a characterisation to be an essential starting point from which to induce a set of comprehensive P&A actions.

The model proposed embraces the notions of openness with regard to HLBSs described in section 3.3.1. However, it attempts to extend these notions by formally describing a HLBS in which:

1. The form and type of formal texts (mark-up languages) that may be incorporated is left open to the designer of the hyperpage;

2. The choice of data model and data query language used to specify and implement the retrieval of content and presentation details of a hyperpage is unconstrained;

3. The formal semantics of hyperpage specifications and personalisation requests are described at a level of abstraction that allows these to be modelled and implemented in a variety of ways using a variety of implementation languages;
4. No constraints are placed on the data, user-interface and possibly the link servers that can be used to serve a HLBS. Data servers are only required to receive requests for data (written in a language of the designer's choice) from the HLBS proper and to serve the HLBS with the requested content. User interfaces are only required to issue requests for hyperpages and to render their results;

5. No predefined structure is assumed for the implementation of links and all link types described in section 3.1.2 may be included;

6. Several Mark-up Languages have been proposed, in particular for the WWW. These languages are explicit document structure mapping tools. Such tools allow for identification and then mapping of meaningful document structure by functions that denote the semantic purpose of the document structure. Although these mark-up languages provide functionality similar to that of the H-region of the proposed model and some do contain constructs to allow the selective inclusion of content and presentation details, none was primarily designed for the personalisation of hyperlink-based interaction. Furthermore, the proposed model supplements its definition of hyperpage specifications found within the H-region with a formal definition of personalisation requests that allow all decisions taken by the designer of a hyperpage to be overridden by the user.

This chapter has described related work on the characterisation of PA-HLBSs in order to highlight similarities and contrasts between the contributions reported in this dissertation with those of other researchers. Although not comprehensive in scope, this chapter has identified many representative examples of hyperlink-based models and systems that relate to the work reported in this dissertation.

Chapter 9 concludes this dissertation by summarising the research contributions made, highlighting conclusions that may be drawn and indicating possible future directions resulting from the approach taken to modelling PA-HLBSs.
Chapter 9
Contributions, Future Directions and Conclusions

This chapter concludes this dissertation by summarising the research contributions made, indicating possible future directions for extending this research and highlighting conclusions that may be drawn from the approach taken to modelling PA-HLBSs.

9.1 Contributions

The following section details the main contributions reported in this dissertation.

The major contribution that this research aims to make to the field of PA-HLBSs research is a comprehensive formalisation of personalisable, adaptive hyperlink-based interaction. The particular attraction of such an approach is that P&A actions can be studied with greater conceptual clarity than is possible by technology-driven experimentation. Furthermore, the formalisation proposed is cast at a level of abstraction above that of concrete systems exploring current technologies, thereby allowing for a principled exploration of specific P&A issues.

The contribution made by this research is unique, as far as the author is aware, in formally modelling a rich set of abstract user-initiated tailoring actions, which enable users to come closer to satisfying their specific, often dynamic, information retrieval goals. The research has shown one approach to modelling personalisable, hyperlink-based interaction that allows system-initiated tailoring actions (adaptivity) to fit cohesively and non-disruptively with user-initiated ones (personalisation).

In this dissertation, system-initiated tailoring has been modelled in a manner that allows it to be, in principle, as expressive as user-initiated tailoring, requiring no other technologies than those involved in user modelling and in decision making from a user model.

In this dissertation, an architecture and decision making algorithm has been proposed that depicts one approach to add, in a non-disruptive manner, adaptive capabilities to the model of user-initiated personalisation proposed in chapter 5.
The approach centres on an adaptation function comprised of:

1. An active rule program composed of a set of active rules that when consulted act as an inference mechanism to determine what is to be adapted, when and how;

2. A database of hyperpages and hyperpage annotations (i.e., the hyper-library) that are the subject of the adaption process;

3. A knowledge base comprised of a goals-library and a history-library;

4. A monitoring system responsible for detecting that a well formed personalisation request has been triggered.

This function implements an inference engine over a decision theory (i.e., a theory as to which actions are more likely to yield the greatest amount of benefits given some accumulated knowledge of past interactions). The actions which the inference engine is in charge of suggesting are P&A actions as defined for user-initiated personalisation.

This dissertation also reports the development of a personalisable HLBS, PAS, that embodies the main concepts underlying the abstract model of personalisable, adaptive hyperlink-based interaction described.

The aim of PAS is to allow for a principled, systematic empirical study to be carried out into the effects of P&A actions. It should be noted that the PAS system is only a “proof of concept” and is intended as a test-bed and demonstrator for the results reported in this dissertation.

9.1.1 Motivation Revisited

The research reported in this dissertation is motivated by the view that merely providing users with browsing facilities does not realise the potential effectiveness of HLBSs in many important information retrieval tasks.

The research reported proposes one approach to overcoming this impediment by extending HLBSs with formally defined P&A actions that effect a complete transfer of ownership from former designers of the hyperdocument to each of its users. This enables users to redesign a hyperdocument according to their specific information goals and histories.

The research is specifically motivated to discover answers to the following fundamental questions:

1. Which are the emergent properties of HLBSs? Equivalently, what is the scope of P&A in HLBSs?
2. Which P&A actions *could* be made available to users? Equivalently, what *descriptive stance* should be taken with respect to P&A actions in HLBSs?

3. Which P&A actions *should* be made available to users? Equivalently, what *prescriptive stance* should be taken with respect to P&A actions in HLBSs?

A further motivating factor for the research reported in this dissertation is the need to address the classical HLBSs issues of cognitive overload and disorientation. It is proposed that by transferring ownership of a hyperdocument from designer to users allows for alterations to the content, rendering and navigational properties of a hyperdocument so as to minimise the levels of cognitive overload and disorientation suffered in navigating the information space made available by the HLBS.

### 9.1.2 Challenges Revisited

The first challenge which the research aimed to address was how to characterise a formal, conceptual model of core hyperlink-based functionality which assumes, but is not determined by, data or user-interface services. Such a characterisation was required to precisely identify the emergent properties of HLBSs.

The second challenge which the research reported aimed to address was how to model, at a suitable level of abstraction, the space of possibilities for P&A actions that could be made available to users of PA-HLBSs. Such a model should characterise the notion of a “transfer of ownership” and should not be technology-driven.

The choice of P&A actions (i.e., the space of possibilities for P&A actions) should fall out from this abstract model. It is envisaged that by modelling the space of possibilities for P&A actions, research can be conducted which will allow the testing, in a principled manner, of the hypothesis that HLBSs that provide P&A features are more effective in information retrieval tasks than those that do not. Furthermore, such research, it is hoped, may highlight those P&A features that are more effective than others.

### 9.1.3 Technical Aims Revisited

The central questions which this dissertation aims to answer are; what is the scope of P&A in HLBSs, and which P&A actions could be made available to users?

The ultimate goal is to complement existing theories and practices of HLBSs research with an abstract formal characterisation of, and an investigation into, personalisable, adaptive hyperlink-based interaction.
The research reported also aimed to address the following derived problems:

1. How to model interaction with a hyperdocument?
2. How to transfer ownership of the process of interaction with a hyperdocument from the designers to its users?
3. How to make design decisions more explicit?
4. How to make design decisions revisable?
5. How to distinguish, model and implement non-tailorable and tailorable hyperdocuments?
6. How to distinguish, model and implement personalisation and adaptivity?

To date, the predominant approach to providing responses to these problems has been to choose a tailoring action and then implement it using a new or existing system. Although such an approach has given valuable insights into how P&A may be implemented and in a relatively few cases their effects, such an approach has often meant that it difficult to generalise and gain insights into any underlying principles.

Furthermore, the motivation and design decisions made in the process of implementing tailoring actions are often difficult to phrase and therefore revise. This approach has resulted in notions of personalisation and adaptivity not being clearly distinguishable, and has resulted in some degree of conceptual muddle, resulting in the drawback of making it difficult to analyse and measure their presumably distinct, independent effects.

9.1.4 Proposed Solution

The research reported in this dissertation responded to the challenges and questions outlined above by:

1. characterising a core of hyperlink-based functionality viewed as a client technology;
2. viewing tailoring actions as ranging over entities within this core and as effecting the transfer of ownership from designers of HLBSs to their users;
3. using formal modelling techniques to make design decisions explicit and therefore revisable;
4. modelling personalisation as user-initiated tailoring of hyperdocuments;
5. modelling adaptivity as system-initiated tailoring;
6. developing a prototype PA-HLBSs based on these models that can be used as a basis to analyse and measure their independent and compound effects.

It was hoped that the methodological procedure of isolating the scope for P&A, defining it formally and then inducing from that formalisation a set of P&A actions, will be a contribution that can be used in other settings, under different assumptions and using alternative conceptualisations of HLBSs.

9.1.5 Summary of Contributions

This research contributes an abstract model of personalisable, adaptive hyperlink-based interaction. The main contributions are, firstly, the definition of an abstract model of core hyperlink-based functionality, and secondly, the definition of an abstract model of personalisability extending the core model. Furthermore, an architecture is drawn which indicates how adaptivity fits into this abstract model and what broad lines its realisation would take.

In this research, the approach taken was to formally model a core of hyperlink-based functionality, referred to as the H-region (see chapter 4). Within the H-region, hyperpages are modelled as formal specifications and a formal language is defined for this purpose. The semantics of hyperpage specifications are defined with reference to a formal abstract machine whose operation and instruction set is specified.

The model of hyperlink-based interaction defined by the H-region provides one answer to the question of what the emergent properties of HLBSs are. It qualifies as one such answer insofar as it does not model any capability that might be inherited from server technologies.

The main contribution of the H-region is to provide a characterisation of core hyperlink-based functionality explicitly, insofar as the emergent properties of the class of HLBSs defined is unambiguously set in contrast to the properties that such systems can simply inherit from other technologies.

The model defined in chapter 4 has the following characteristics:

1. The conceptual framework of the H-region allows for the specification of independently defined and independently accessible units of contents (chunks). The presentation of these units of information is separated for their content and any appropriate presentation mark-up language may be incorporated seamlessly;

2. All forms of data and media may be referenced using hyperpage specifications written in the formal grammar defined for hyperpages;
3. The model allows for the inclusion of freely defined links between freely defined units of information;

4. The model proposes an abstract machine that realises the static and dynamic generation of hyperpages.

In summary, the H-region aims to clearly delineate the scope of P&A actions as a core of functionality which can be realised as a client technology of content and presentation servers.

In chapter 5, an abstract, additive model for personalising hyperlink-based interaction is described, referred to as the P-region. The P-region provides one answer to the question of which P&A actions could be made available to users. It qualifies as one such answer insofar as it indicates how, given a conceptual framework such as that which underlies the H-region, it is possible to devise a formal language for personalisation whose effects fall out as a consequence of that very conceptual framework being adopted.

The P-region defines a set of annotation possibilities and P&A actions that are induced from the formal definition of hyperpage specifications defined by the H-region. These annotation possibilities and P&A actions enable all design decisions realised as hyperpage specifications to be revised by users.

Personalisation has been modelled as the user-initiated process of annotating and rewriting a hyperpage specification into a version thereof that is associated with the user who took that action.

Hyperpage annotations and personalisation requests are modelled as formal specifications and a formal language has been defined for this purpose. The semantics of personalisation requests are expressed using relational algebra with assignments.

When personalisation functionality is layered over the core, a designer can annotate a hyperpage in preparation for differences in users’ goals and histories. A user can request to personalise not only such annotations, but the hyperpage specifications as well. Personalisation requests allow users to specify which hyperpages are to be personalised and how they should be transformed. The P-region provides the functionality needed for P&A actions without disrupting and, in fact, relying on the functionality of the H-region remaining intact and unaltered.

The model defined in chapter 5 has the following characteristics:

1. All decisions designers of hyperdocuments make with regard to content, structure and presentation of units of information can be the subject of P&A actions;

\[1\] In appendix 5.5, an alternative set-theoretic representation of the semantics of personalisation requests is given for the reader that is better acquainted with this formalism.
2. The conceptual framework of the P-region allows for a complete transfer of ownership of the hyperdocument from its designers to its users;

3. All P&A actions are modelled explicitly and are formally defined;

4. All P&A actions modelled by the P-region are consistent in nature and are repeatable and revisable by designers and users.

In summary, within the P-region what is being tailored are unique, distinctive properties of hyperlink-based interaction, in contrast to emergent properties such as those that simply arise from the coupling of database and user interface technologies in a particular manner using some particular protocol.

In chapter 6, an abstract, additive model of system-initiated, personalisable hyperlink-based interaction is described, referred to as the A-region. The A-region provides one answer to the question of how system-initiated P&A actions could be made available to users. It qualifies as one such answer insofar as it indicates how, given a conceptual framework such as those that underlie the H- and P-regions, it is possible to devise a simple architecture to model system-initiated tailoring, namely, adaptivity.

It is complete, in the sense that it is expressive enough to illustrate how adaptivity can be modelled as a coherent and consistent extension of the H- and P-regions. Furthermore, it shows that using the conceptual frameworks of the H- and P-regions, adaptivity can be as expressive as personalisation. This is an important result, insofar as that to the author’s knowledge, no system proposed has been able to implement this level of functionality and no model has shown this to be possible.

Adaptivity is modelled as the process of allowing the system to take the initiative in personalisation actions, in the light of the system’s inference of a user’s information goals and history. When adaptivity functionality is layered over the H- and P-regions, both users and designers can define strategies as to when the system should take the initiative and actively tailor the interaction to a user.

In chapter 6, an architecture is proposed for an adaptive function, the major components of which are: a database of hyperpages and hyperpage annotations, a knowledge base of user goals and histories and an active rule program comprised of a set of active rules over an active database. It is shown that the active rule program can be extended to accommodate rules written using any appropriate event, condition and action specification languages.

The model proposed in chapter 6 has the following characteristics:

1. Within the A-region, the system tailors interaction by issuing personalisation requests as defined by the P-region that in turn have been induced from a core of hyperlink functionality, the H-region;
2. All decisions designers and users of hyperdocuments make with regard to content, structure and presentation of units of information are able to be the subject of P&A actions issued by the system;

3. Within the A-region, adaptivity is modelled separately from personalisation. This approach allows a simple representation of what adaptivity is and how in particular implementations, appropriate user models and decision making algorithms can be used to realise P&A actions;

In summary, within the A-region the notion of adaptivity is shown to be system-oriented. It is rooted in a mechanism that allows the system to detect events, make conjectures and issue commands. It is not specific to HLBSs but can be modelled, for example, using established active database theories and can be implemented using established database management systems.

To summarise, the model proposed in this dissertation contributes two answers to the three basic P&A questions proposed in 9.1.1:

1. What is the scope of P&A (both user and system-initiated) in HLBSs?

A view of a core of hyperlink-based functionality is adopted and formalised. This view presupposes a variety of servers to which HLBSs are only loosely coupled. It is shown how this functionality emerges from a formal definition of hyperpages. It is then proposed that it is the formal elements in this formal definition that are within the scope of P&A actions. When such formal elements are personalised the composition of specifications into renderable documents effects the transfer of ownership;

2. Which P&A actions could be made available to users?

It is proposed that all formal elements in the formal definition of a hyperpage are within the scope of P&A actions. This view essentially means that every decision made by the designers can be overridden by a P&A action. Equivalently, every hyperpage can be entirely redesigned by the set of P&A actions induced from the proposed abstract model. In this dissertation, it is argued that empirical studies should be carried out before any prescriptive stance is adopted.

The model of personalisable, adaptive hyperlink-based interaction:

1. is an abstract model, as many steps removed from concrete implementations as necessary to allow a systematic, exhaustive investigation of P&A issues in HLBSs;

2. is an open model, insofar as HLBSs are viewed as clients of a variety of servers and in particular of data and user-interface servers;
3. models personalisation as a transfer of ownership of hyperdocuments, from designers to users;
4. induces a set of personalisation actions from a formal definition of the hyperdocuments they act upon;
5. can express, to the authors’ knowledge, all personalisation actions proposed in the literature (see [Brusilovsky, 1996b] for a comprehensive review).

The remaining question, which P&A actions should be made available to users, will require empirical studies conditional on the answers addressed by the research reported here.

**PAS: A Personalisable Hyperlink-Based System**

This dissertation reports on the development of a personalisable, HLBS referred to as PAS. The main contribution of PAS is to substantiate the claim that the abstract approach taken allows not only for a greater understanding of what personalisation and adaptivity mean in the context of HLBSs, but also how the model may aid the design of such systems.

The PAS system consists of two major components: an instantiation of the H-region as a WWW-based application for realising dynamically generated hyperdocuments; and an instantiation of the P-region as a parser for personalisation requests. The PAS system has been implemented using the client-server technologies for WWW-based data retrieval and presentation. The system, together with the parser implementation for the formal languages described above, aims to demonstrate the provability of the proposed model.

The PAS system will be used in future work as a foundation for a systematic investigation of the effects of P&A actions.

**9.1.6 Applications for Research**

It is realised that the research reported in this dissertation is not, by itself, a complete solution to the many issues concerned with modelling PA-HLBSs. However, it is envisaged that it may be of significant value to the following application areas.

At present, there are many efforts to design and implement WWW-based applications for a variety of application domains (e.g., electronic commerce, advertising, customer support). In particular, there is a trend towards the design of WWW-based applications that not only provide information to users, but also customise this information on an individualised basis.

A particular application domain of the work reported in this dissertation is that of business-to-customer electronic commerce. Companies wishing to participate in electronic markets
are faced with the challenge of how to present their goods and services to customers without the benefits of face-to-face communication. In traditional commerce, sales-people are often used to provide customised information to potential customers on goods and services available. In electronic markets such face-to-face communication is often not possible. However, customised information could be made available if electronic commerce applications possessed the functionality necessary to personalise the interaction process that takes place between the system and its potential customer.

It is hoped that the model reported in this dissertation may be used by the designers of electronic commerce applications as a framework for implementing personalisation and adaptation features within such systems.

The majority of PA-HLBSs proposed to date are to be found in the area of education. The P&A actions proposed herein could potentially be used to address a number of issues concerned with educational HLBSs. Students will generally have varying degrees of knowledge of a subject domain, furthermore, such knowledge is expected to increase over time as new knowledge is acquired. The P&A actions described in this work may be used to restrict the type and amount of information contained within a hyperdocument to that which is deemed appropriate for an individual student at a particular point in time. P&A actions may also be used to limit the navigational possibilities available to a student when interacting with an educational HLBS. P&A actions may be used to restrict navigational possibilities to a subset that either a teacher and/or a student deems to be appropriate. The annotation possibilities made available by the proposed model may be used to provide navigational help to students while they are interacting with an educational HLBS. In particular, in the proposed model, it is possible to specify alternatives to a presented link and even to specify that some other link should be traversed before the link is presented.

The annotation possibilities proposed by the model are cast at a level of abstraction that allows further extensions to the set of attributes defined. Furthermore, it is possible to redefine this set to an appropriate set of attributes in particular contexts. A fixed set of attributes is modelled for hyperpages to illustrate their use. However, it is hoped that from such an illustration alternative strategies can be devised by educational HLBSs developers to apply the opportunities afforded by the annotation of hyperpages in an educational context.

A further application of the annotation possibilities proposed is the use of annotations to enforce group level filtering of content with a HLBS. Many organisations are now employing Internet-based technologies to provide internal information within their organisations. Such applications are broadly referred to as Intranet applications. Here, the specification of annotations provided could be used as a framework for the development of Intranet-based management information systems that provide different levels of information to different groups within an organisation.
The proposed model may also be applied to the development of systems for managing personalised views and information spaces. Wide scale, public HLBSs such as the WWW provide users with a potentially unlimited amount of information. It can be concluded that such systems will require tools and techniques to personalise the information they make available. The P&A actions formally defined may be used to aid the design of tools and techniques which may be employed to realise personalised views of information held in, for example, the WWW.

Other potential application areas for the research reported here include the design and implementation of: on-line help systems; on-line information systems; and institutional information systems.

9.2 Future Directions for Research

The following section highlights several possible directions for future PA-HLBSs research that have become apparent whilst conducting the research reported in this dissertation.

9.2.1 Structural Hyperlink-Based Computing

Although OHLBSs researchers have made various proposals as to the range of services and protocols that HLBSs should provide [Davis and Lewis, 1996, Wiil and Grønbæk, 1997], the exact number and type of services must be resolved for further progress to be made [Nürnberg and Leggett, 1998].

In this dissertation, an alternative view of what constitutes a HLBS is presented (see subsection 3.5) and the terms hypermedia and hypertext are generalised to the term hyperlink. The reasons for this are described in 2.1.2. A HLBS is viewed as a structural component whose functionality provides the ability to interpret a formal specification, retrieve content (held locally and remotely) and to compose that content into a formal text that some presentation tool (browser or otherwise) can render and then present to the user.

From this viewpoint the following tentative conclusions can be drawn:

1. Although a HLBS requires content, the management of this content and methods for retrieving this content are outside the scope of the HLBS. There are already established technologies for the management and retrieval of content (i.e., DBMSs, file systems) and these technologies should be employed by HLBSs to provide this functionality;

2. The form of links (which are just further content) and the functionality they provide
should also be viewed as outside the scope of the HLBS. The potentially unlimited ways in which links may be realised and the many different contexts in which they may be deployed, lends weight to the argument that HLBSs should not place constraints on their form or the functionality they can provide;

A link may be viewed as simply content by a HLBS, it is the rendering of a well formed formal text (hyperpage) that realises this content into executable regions on a screen, that may be activated and used as access mechanisms to further content. For example, if we take a HTML formal text that contains the following content `<a HREF="http://www.mypages/index.html">mypage</a>` and this content is rendered by a HTML compliant browser, then it will be realised as a link to some content. Conversely, if a HTML formal text has content of this form `<a HREF="http://www.mypages/index.html">mypage</a>` (i.e., where the second double quotation mark is missing) it will be realised as a string of text. The difference between the two is how the designer specifies the original formal text. This example illustrates that in a WWW environment, it is the designer, not the HLBS that is responsible for the specification of the form links and the functionality they provide;

3. Many server technologies may be used to present hyperlinked information to users. At present, the most popular technologies are WWW browsers. The functionality provided by these WWW browsers is evolving rapidly and, furthermore, these WWW browsers are often used as references to, or containers for, other presentation technologies such as word processors, document readers and spreadsheets. For example, it is becoming increasingly common to observe WWW pages that present content using distinct presentation technologies to render particular content (i.e., MS Excel, to render spreadsheets, MS Word to render word processed documents and Adobe Acrobat to render Adobe PDF files). What these observations suggest is that the functionality that such technologies provide should also be distinct from the HLBS itself. The HLBS should employ these technologies when required;

4. OHLBS research is based on the view that most applications are hyperlink unaware (i.e., have no notion of linking) but may benefit from such functionality. It was originally argued that a central role of OHLBSs research is to devise systems to enable such applications to become hyperlink aware. However, an ever increasing number of applications may be described as hyperlink aware [Grauer, 1997]. Although this new generation of applications are hyperlink aware, structure management (i.e., management of the hyper-network) is still largely ad-hoc. What these observations suggest is that the focus of OHLBSs research should now fall on devising effective support mechanisms for structurally-oriented computing.

The question therefore arises as to what is the functionality provided by the HLBS itself.

In this dissertation, a HLBS is a composer from specifications, it is not an application, and
is not a database. A model is proposed which aims to address the issue of how to formally model a tailorable hyperlink structure which can effectively support a structurally-oriented hyperlink computing environment. It is a structural component that provides the functionality to interpret and then compose a specification into a formal text. Although one proposal is put forward for the form of this structural component, research should be carried out into alternative characterisations of this structural component, so that comparisons can be made and a consensus of opinion can be reached regarding its form and operation.

9.2.2 Defining the PA-HLBSs Research Space

From the raft of proposals for, and implementation of, PA-HLBSs, it can be seen that personalisation and adaptation are viewed in many different ways and the levels of functionality offered vary widely. There is no consensus as to what constitutes adaptive hypermedia and no general accepted scope for P&A actions in HLBSs. In many of the implementations presented, it is not clear that what is being personalised or adapted is unique to HLBSs and cannot be easily duplicated by other classes of systems. Clearly, a minimal requirement of PA-HLBSs is that they personalise and adapt functionality that is unique to HLBS. It can be argued that not to do so would mean that PA-HLBSs are only providing an alternative solution to a problem that has already been largely solved. For example, the retrieval of content from a database using a DML, DQL to create personalised views of the content held within the database.

Therefore, a significant area of future research must be the establishment of a consensus of the space of possibilities for P&A actions. This dissertation proposes one approach to defining this space of possibilities. However, it is acknowledged that for a consensus to be arrived at, different conceptual frameworks that induce different views of this space of possibilities will have to be proposed and then contrasted with that reported here.

9.2.3 HLBSs and Active Databases

In this dissertation, an architecture and decision making algorithm has been proposed that depicts one approach to modelling and then implementing adaptivity within HLBSs, the A-region. The approach centres on an adaptation function whose specification draws upon research into active database theory. Although this approach clearly shows how active databases may be used to realise adaptation of hyperlink-based interaction, further work on formally understanding and then integrating active databases into HLBSs is required. Work is currently being conducted in this area and the PAS system is presently being extended to incorporate the functionality described by the A-region.

Furthermore, it is also important that some principles regarding what functionality any
adaptive function should provide, and protocols for the use of any such functionality must be established. It is clearly not enough to simply implement in a “closed manner” an adaptive function. Its works and interfaces should be “open to scrutiny” and should be able to be compared and contrasted with that of others.

The architecture described here shows one approach to add, in a non-disruptive manner, adaptive capabilities to the model of user-initiated tailoring proposed in chapter 5. This is only one approach, conceivably there are many others which should be explored and then compared and contrasted. Furthermore, the architecture proposed essentially treats the adaptive function as a “black box” with few constraints on the functionality it should provide. The validity of this approach will require verification via research activities aimed at testing the suitability of the proposed architecture for including alternative characterisations of adaptive functions.

9.2.4 Principles for Personalisation and Adaptation

PA-HLBSs research has yet to yield any consensus with regard to underlying principles for personalisation, a taxonomy or even a set of terms and concepts to be used. The work reported in this dissertation aims to promote further research in this area. It proposes that:

1. A hyperpage, from the view of a HLBS, is a specification of where and how to source content and how to present that content;
2. A HLBS is a composer of renderable texts from hyperpage specifications;
3. A hyperdocument is a collection of hyperpage specifications;
4. Personalisation is the user-initiated rewriting of hyperpage specifications;
5. Adaptation is the system-initiated rewriting of hyperpage specifications.

Personalisation, whether system or user-initiated, should embody the following principles:

1. Personalisation should, to some degree, represent the measured transfer of ownership of the process of interaction with a hyperdocument from designer to user;
2. All hyperpage design decisions should be able to be the subject of personalisation requests;
3. A model of hyperlink-based personalisation should aim to accommodate all recognised personalisation actions;
4. Personalisation actions should be clearly defined, explicit and capable of being formally understood;
5. All personalisation actions should be consistent, repeatable and revisable;
6. All personalisation actions should ultimately be subject to empirical tests for effectiveness gains.

These definitions and principles are viewed as a starting point. Much work needs to be conducted to establish more precise, and perhaps alternative, definitions and principles. As with most areas of research, PA-HLBSs require a solid theoretical foundation, and at present this is not the case. A major challenge for HLBSs researchers must therefore be to build this foundation.

9.2.5 Large Scale Empirical Research

It is important that large scale empirical studies into the effects of P&A be carried out so that a prescriptive stance can be established regarding which P&A actions should be made available to users in particular contexts and for particular applications. It is felt that this question is, in part, conditional on the answers addressed by the research reported in this dissertation.

Although at present the PAS personalisable HLBS described in chapter 7 does not possess the functionality required to realise adaptivity. Future work on PAS will be concerned with how to implement this functionality so empirical testing of the independent effects of user-initiated and system-initiated P&A actions can be carried out. Such tests, it is hoped, will go some way to providing a prescriptive stance as to which P&A actions are more or less effective in different contexts.

9.3 Research Conclusions

The body of research into HLBSs and the research reported in this dissertation indicates that a HLBS should be viewed as a structure that, by definition, must aim to fit within the wider computing environment. From this viewpoint, it can be concluded that PA-HLBSs research should be concerned with personalising and adapting the behaviour that such structures can exhibit and the functionality they provide.

It is necessary for PA-HLBSs researchers to form some consensus on what should be the subject of P&A actions. Clearly, it is more advantageous to direct HLBSs personalisation and adaptation research at issues that are unique, identifying properties of HLBSs as opposed to those properties which HLBSs inherit as a matter of course.

Although it is necessary to build PA-HLBSs to understand how they may be realised, it is also important to formally understand the underlying principles of the behaviour these systems seek to exhibit.
It is only through such a formal understanding that underlying principles and practices can emerge in a well formed manner. Furthermore, such an approach may bring some clarification to issues of interoperability and interchangeability between different PA-HLBSs.

It is hoped that the formal approach taken of modelling PA-HLBSs at a level of abstraction above that of concrete systems, may clarify what it is within HLBSs that should be subject to P&A actions and what actions are possible.

Such clarification, it is hoped, can be used as a gateway for a principled evaluation of the effects of different P&A actions in different contexts. It may also help to provide a conclusive answer to the prescriptive question of which P&A actions should be made available to different types of user in different contexts.

Finally, it is hoped that the research reported herein has provided the beginnings of a foundation for devising a set of underlying principles and a taxonomy for PA-HLBSs that can be used in a wider context.
Appendix A
Formal Semantics of Personalisation Requests using Set-theoretic Expressions

This appendix details an alternative representation of the formal semantics of personalisation requests (given in section 5.5.3) using set-theoretic expressions. This is provided as an alternative for the reader that is better acquainted with this formalism.

A.1 The Meaning Function

Given the successful parsing of a personalisation request whose output is a term representation of the syntax tree of the personalisation request, denoted by $T$, the semantics of $T$, denoted by $[[T]]$, is recursively defined in subsection A.1.1. Figures A.1 to A.6 define functions which are referenced to in the definition of the recursive function for personalisation requests.

The meaning of an action list, $AL$ is defined as a function $[[AL]]$ which takes a parameter, $P$ of type page. $[[AL]](P)$ is written to denote the applying of $[[AL]]$ to the parameter $P$.

Notational Conventions and Assumptions

In the definitions that follow, a shift, in the grammar for hyperpages and hyperpage annotations, is assumed to be represented as a signed integer. These definitions can be extended to include the case where a shift is the token $all$ but to aid clarity, this is not discussed here. Section A.1.2 outlines how such extensions can be made. Furthermore a shift value denoted by $shift$, may possibly be sub- or superscripted.
“E1 ; E2” is written to mean “do E1 and then do E2”.

“R’ = S” to indicate changing the relation named R to be S.

“ foreach P ∈ S do E endfor” is written to mean “perform E for each P in the finite set S”.

“*” denotes any value in the domain of the attribute in its position in a relation.

A.1.1 Meaning: Personalisation Request

Given a set of pages, personalisation_request selects those pages in a specified action-scope and applies a set of personalisation actions to this subset.

if T = 'personalisation_request(AS,AL)'
    then
        [[T]] = [[foreach P ∈ [[AS]] do
                   [[AL]](P)
                   endfor]]

Meaning : Action Scope

Given the token select_page_if and a condition, action_scope selects those pages that satisfy the specified condition.

if T = 'action_scope(select-page-if,C)'
    then
        [[T]] = [[C]]
Meaning : Selection Conditions

Given an atomic selection condition, selection condition tests each page in the action scope to see whether or not the atomic selection condition is satisfied. A selection condition may be proceeded by any of the tokens not, and, or, and these are defined in the standard way. The tokens and, or each require two atomic selection conditions as parameters.

if \( T = \text{\texttt{\textbackslash''selection\_condition(atomic\_selection\_condition(C))}} \) then
   \[
   \llbracket T \rrbracket = \llbracket C \rrbracket 
   \]

if \( T = \text{\texttt{\textbackslash''selection\_condition(not,selection\_condition(C))}} \) then
   \[
   \llbracket T \rrbracket = \{ \text{\texttt{\textbackslash''page}}} | (\text{\texttt{\textbackslash''page}}, \ast, \ast) \in \text{\texttt{\textbackslash''PAGE}} \} \setminus \llbracket C \rrbracket 
   \]

if \( T = \text{\texttt{\textbackslash''selection\_condition(and,selection\_condition(C_1),selection\_condition(C_2))}} \) then
   \[
   \llbracket T \rrbracket = \llbracket C_1 \rrbracket \cap \llbracket C_2 \rrbracket 
   \]

if \( T = \text{\texttt{\textbackslash''selection\_condition(or,selection\_condition(C_1),condition(C_2))}} \) then
   \[
   \llbracket T \rrbracket = \llbracket C_1 \rrbracket \cup \llbracket C_2 \rrbracket 
   \]

Meaning : Atomic Selection Condition True

Given the token \texttt{true}, an atomic selection condition selects all pages.

if \( T = \text{\texttt{\textbackslash''atomic\_selection\_condition(true)}} \) then
   \[
   \llbracket T \rrbracket = \{ \text{\texttt{\textbackslash''page}}} | (\text{\texttt{\textbackslash''page}}, \ast, \ast) \in \text{\texttt{\textbackslash''PAGE}} \} 
   \]

Meaning : An Atomic Selection Condition on a Page

Given an atomic selection condition on page whose parameter is the token \texttt{page}, and a containment expression, \( EXP \), whose parameters are the token \texttt{contains}, and a regular expression, atomic selection condition selects those pages containing the specified regular expression.

if \( T = \text{\texttt{\textbackslash''atomic\_selection\_condition(atomic\_selection\_condition\_on\_page
   (page, containment\_expression(contains,EXP))))}} \) then
   \[
   \llbracket T \rrbracket = \{ \text{\texttt{\textbackslash''page}}} | (\text{\texttt{\textbackslash''page}}, \ast, \ast) \in \text{\texttt{\textbackslash''PAGE}} \land EXP \prec \text{\texttt{\textbackslash''page}} \} 
   \]

216
Meaning: Atomic Selection Condition on a Page Part

Given an atomic_selection_condition on_page_part whose parameters are a sequence of values and tokens denoting a specified page-part and a containment_expression whose parameters are the token contains, and a regular expression, EXP, atomic_selection_condition selects those pages whose specified page-part contains the specified regular expression.

\[
\text{if } T = \text{atomic_selection_condition(atomic_selection_condition_on_page_part}
\]
\[
\text{page_part((relevant_chunk(shift, chunk))},
\]
\[
\text{containment_expression(contains.EXP))')}
\]
\[
\text{then }
\]
\[
\llbracket T \rrbracket = \{\text{page}|\langle \text{page}, \text{shift}, \text{chunk} \rangle \in \text{PAGE} \land EXP \subset \text{chunk}\}
\]
\[
\text{if } T = \text{atomic_selection_condition(atomic_selection_condition_on_page_part}
\]
\[
\text{page_part(relevant_chunk(shift_1,\text{chunk}},
\]
\[
\text{(relevant_multi_part(shift_2,entry-point))})},
\]
\[
\text{containment_expression(contains.EXP ))')}
\]
\[
\text{then }
\]
\[
\llbracket T \rrbracket = \{\text{page}|\langle \text{page}, \text{shift_1}, \text{chunk} \rangle \in \text{PAGE} \land
\]
\[
\exists \text{entrypoint, entrypointset.}\langle \text{chunk, entrypointset, *,*,*} \rangle \in \text{CHUNK} \land
\]
\[
\langle \text{entrypointset, shift_2, entrypoint} \rangle \in \text{ENTRYPOINTSET} \land EXP \subset \text{entrypoint}\}
\]
\[
\text{if } T = \text{atomic_selection_condition(atomic_selection_condition_on_page_part}
\]
\[
\text{page_part(relevant_chunk(shift,\text{chunk}},(relevant_single_part(c-spec))},
\]
\[
\text{containment_expression(contains.EXP ))')}
\]
\[
\text{then }
\]
\[
\llbracket T \rrbracket = \{\text{page}|\langle \text{page}, \text{shift}, \text{chunk} \rangle \in \text{PAGE} \land
\]
\[
\exists \text{c-spec.}\langle \text{chunk, *,c-spec,*,*} \rangle \in \text{CHUNK} \land EXP \subset \text{c-spec}\}
\]
\[
\text{if } T = \text{atomic_selection_condition(atomic_selection_condition_on_page_part}
\]
\[
\text{page_part(relevant_chunk(shift,\text{chunk}},(relevant_single_part(r-spec))},
\]
\[
\text{containment_expression(contains.EXP ))')}
\]
\[
\text{then }
\]
\[
\llbracket T \rrbracket = \{\text{page}|\langle \text{page}, \text{shift}, \text{chunk} \rangle \in \text{PAGE} \land
\]
\[
\exists r\text{-spec.}\langle \text{chunk, *,*,r-spec,*} \rangle \in \text{CHUNK} \land EXP \subset \text{r-spec}\}
\]
\[
\text{if } T = \text{atomic_selection_condition(atomic_selection_condition_on_page_part}
\]
\[
\text{page_part(relevant_chunk(shift_1,\text{chunk}},(relevant_multi_part(shift_2,entry-point))})},
\]
\[
\text{containment_expression(contains.EXP ))')}
\]
\[
\text{then }
\]
\[
\llbracket T \rrbracket = \{\text{page}|\langle \text{page}, \text{shift_1}, \text{chunk} \rangle \in \text{PAGE} \land
\]
\[
\exists \text{exitpoint, exitpointset.}\langle \text{chunk, exitpointset, *,*,*} \rangle \in \text{CHUNK} \land
\]
\[
\langle \text{exitpointset, shift_2, exitpoint} \rangle \in \text{EXITPOINTSET} \land EXP \subset \text{exitpoint}\}
\]
Meaning: Atomic Selection Condition on Note

Given an atomic_selection_condition_on_page_part whose parameter is a value, shift, which denotes a shift, the token note and a containment expression whose parameters are the token contains, and a regular expression, EXP, atomic_selection_condition selects those pages with associated notes that contain the specified regular expression.

if T = 'atomic_selection_condition(atomic_selection_condition_on_page_part (page_part(relevant_note(shift,note))), containment_expression(contains,EXP )))'
   then
   \([T] = \{ \text{page} | (\text{page}, \ast, \ast) \in \text{PAGE} \land \exists \text{note}. (\text{note}, \ast, \text{chunk}, \ast, \ast, \text{shift}, \ast, \text{n_string}, \ast) \in \text{NOTE} \land EXP \prec \text{n_string} \}\)

Meaning: Action List

Given a list of actions to be performed on a hyperpage or its associated hyperpage annotation, action_list applies this list of actions to that hyperpage or its associated hyperpage annotation action_list takes a parameter P that denotes a page.

if T = 'actionlist(A,AL)'
   then
   \([T]_P = \{ \text{A} \}_P \circ \{ \text{AL} \}_P \)

Meaning: Action

Given the token an_then_do and a list of actions to be performed on hyperpage annotations, action applies this list of actions to a selected page in the action scope denoted by P. Given the token hp_then_do and a list of actions to be performed on hyperpages, action applies this list of actions to a selected page, P.

if T = 'action(ann_then_do,ATD)'
   then
   \([T]_P = \{ \text{ATD} \}_P \)

if T = 'action(hp_then_do,HTD)'
   then
   \([T]_P = \{ \text{HTD} \}_P \)
Meaning: Annotation Update with a Query on the Environment Condition

Let true and false denote the Boolean truth values, then given an update operator whose parameter is the token insert (or the token delete), a condition whose parameters are the token if-not or if and a value, QOE, which denotes the token QUERY-ON-ENVIRONMENT, and a note, whose parameter is itself a note tuple as defined in 5.5.1 and represented here as note, annotation update calls a function InsertNote (or DeleteNote) which takes a parameter, P of type page if the query on the environment is true in the case of the token if, or false in the case of the token if-not.

if \( T = '\text{annotation}_\text{update}(\text{update}_\text{operator}(\text{insert}), \text{condition}(\text{if-not}, QOE), \text{note}(n))' \)
then
\[ \llbracket T \rrbracket(P) = \text{InsertNote}(\text{note})(P) \] if QOE is false

if \( T = '\text{annotation}_\text{update}(\text{update}_\text{operator}(\text{insert}), \text{condition}(\text{if}, QOE), \text{note}(n))' \)
then
\[ \llbracket T \rrbracket(P) = \text{InsertNote}(\text{note})(P) \] if QOE is true

if \( T = '\text{annotation}_\text{update}(\text{update}_\text{operator}(\text{delete}), \text{condition}(\text{if-not}, QOE), \text{note}(n))' \)
then
\[ \llbracket T \rrbracket(P) = \text{DeleteNote}(\text{note})(P) \] if QOE is false

if \( T = '\text{annotation}_\text{update}(\text{update}_\text{operator}(\text{delete}), \text{condition}(\text{if}, QOE), \text{note}(n))' \)
then
\[ \llbracket T \rrbracket(P) = \text{DeleteNote}(\text{note})(P) \] if QOE is true
Meaning: Annotation Update Note

Given an update operator whose parameter is the token `insert` (or the token `delete`) and a note, whose parameter is itself a note tuple as defined in 5.5.1 represented here as `note`, annotation update calls the function `InsertNote` (or `DeleteNote`) which takes a parameter, \( P \) of type page.

\[
\text{if } T = \text{`annotation_update(update_operator(insert),note(note))' if } T = \text{`annotation_update(update_operator(delete),note(note))' then}
\]
\[
\lbrack\lbrack T \rbrack\rbrack(P) = \text{InsertNote(note)(P)}
\]
\[
\text{if } T = \text{`annotation_update(update_operator(delete),note(note))' then}
\]
\[
\lbrack\lbrack T \rbrack\rbrack(P) = \text{DeleteNote(note)(P)}
\]

Meaning: Annotation Update Note Projection

Given a note projection whose parameters are projection operator whose parameter is the token `drop-if` (or the token `retain-if`), and a selection condition on note, annotation update deletes a note from the relation `NOTE` if that note is associated with a chunk found in a page in \( P \) which in the case of drop-if meets the selection condition on note and in the case of retain-if does not meet that selection condition on note.

\[
\text{if } T = \text{`annotation_update(note_projection(projection_operator(drop_if)),)
\]
\[
\text{selection_condition_on_note(CN))'}
\]
\[
\text{then}
\]
\[
\lbrack\lbrack T \rbrack\rbrack(P) = \text{NOTE}'=\text{NOTE} \setminus \{\langle\text{note},*,\text{chunk},*,*,*,*,*\rangle|\langle\text{chunk},*,*,*,*\rangle \in \text{CHUNK} \land \exists \text{chunk}.\langle P,\text{chunk},*\rangle \in \text{PAGE} \land \lbrack\lbrack \text{CN} \rbrack\rbrack(P)\}
\]
\[
\text{if } T = \text{`annotation_update(note_projection(projection_operator(retain_if)),)
\]
\[
\text{selection_condition_on_note(CN))'}
\]
\[
\text{then}
\]
\[
\lbrack\lbrack T \rbrack\rbrack(P) = \text{NOTE}'=\text{NOTE} \setminus \{\langle\text{note},*,\text{chunk},*,*,*,*,*\rangle|\langle\text{chunk},*,*,*,*\rangle \in \text{CHUNK} \land \exists \text{chunk}.\langle P,\text{chunk},*\rangle \in \text{PAGE} \land \lbrack\lbrack \text{CN} \rbrack\rbrack(P)\}
\]
Meaning : Annotation Update Insert Annotation

Given the token annotation, annotation_update inserts an annotation for a page denoted by $P$. If an annotation has previously been inserted into the relation ANNOTATION for a page denoted by $P$ then it is deleted.

if $T = \text{'annotation_update(annotation)'}$
  then
    $[[T]](P) = \text{ANNOTATION}^\prime = \text{ANNOTATION} \setminus \{(\ast, P)\ |
    \langle P, \ast, \ast \rangle \in \text{PAGE}\}$
    $\text{ANNOTATION}^\prime = \text{ANNOTATION} \cup \{(\text{newannotation}, P)\ |
    \langle P, \ast, \ast \rangle \in \text{PAGE}\}$
    where $\forall \langle\text{annotation}, \ast\rangle \in \text{ANNOTATION.\ annotation} \neq \text{newannotation}$

Meaning : Selection Condition on Note

Given an atomic_selection_condition_on_note whose parameter is a value, shift, which denotes a shift, the token note and a containment expression whose parameters are the token contains, and a regular expression, EXP, selection_condition_on_note returns the value true or false if the page denoted by $P$ is associated with at least one note that contains the specified regular expression.

if $T = \text{'selection_condition_on_note(atomic_selection_condition_note(CN))'}$
  then
    $[[T]](P) = [[\text{CN}]](P)$

if $T = \text{'selection_condition_on_note(not,selection_condition_on_note(CN))'}$
  then
    $[[T]](P) = \neg[[\text{CN}]](P)$

if $T = \text{'selection_condition_on_note(and,}
    \text{selection_condition_on_note(CN}_1\text{),selection_condition_on_note(CN}_2\text{))'}$
  then
    $[[T]](P) = [[\text{CN}_1]](P) \land [[\text{CN}_2]](P)$

if $T = \text{'selection_condition_on_note(or,}
    \text{selection_condition_on_note(CN}_1\text{),selection_condition_no_note(CN}_2\text{))'}$
  then
    $[[T]](P) = [[\text{CN}_1]](P) \lor [[\text{CN}_2]](P)$
if \( T = '\text{selection\_condition\_on\_note(atomic\_selection\_condition\_on\_note} \\
\text{\(\text{page\_part(relevant\_note(shift,note)), containment\_expression(contains,EXP)})\')' \)

then

\[
\llbracket T \rrbracket (P) = \exists \text{chunk}. \langle P, *, \text{chunk} \rangle \in \text{PAGE} \land \\
\langle \text{chunk}, *, *, *, * \rangle \in \text{CHUNK} \land \\
\exists \text{note}. \langle \text{note}, \text{chunk}, *, shift, *, n\_string, * \rangle \in \text{NOTE} \land EXP \triangleleft n\_string
\]

Meaning: Hyperpage Structure Update Insert Page Part

Given a \( \text{hp\_structure\_update} \) whose parameters are the token \( \text{insert} \), a \( \text{page\_part} \) whose parameters are a sequence of values and tokens denoting a specified page-part and a \( \text{hp\_constr} \) whose parameter is one or more parts of a chunk as defined in the grammar for hyperpages, \( \text{hp\_update} \) inserts those parts of a chunk into a page denoted by \( P \).

if \( T = '\text{hp\_update(hp\_structure\_update(insert,} \\
\text{\(\text{page\_part(relevant\_chunk(shift,chunk,} \\
\text{(relevant\_multi\_part(shift2,entry-point))))}), hp\_constr(entry-point(entry(uis))))')' \)

then

\[
\llbracket T \rrbracket (P) = \text{ENTRYPOINT}' = \text{ENTRYPOINT} \cup \{ \langle \text{newentry}, uis \rangle \} \\
\text{ENTRYPOINTSET}' = \text{ENTRYPOINTSET} \cup \{ \langle \text{entrypointset}, \text{newshift}, \text{newentry} \rangle | \\
\exists \text{chunk}. \langle \text{chunk}, \text{entrypointset}, *, *, *, * \rangle \in \text{CHUNK} \land \\
\langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE} \} \\
\text{where } \forall \langle \text{entrypoint}, * \rangle \in \text{ENTRYPOINT}. \text{entrypoint} \neq \text{newentry} \land \\
\forall \langle \text{entrypointset}, \text{shift}, * \rangle \in \text{ENTRYPOINTSET}. \text{shift} \neq \text{newshift}
\]

if \( T = '\text{hp\_update(hp\_structure\_update(insert,} \\
\text{\(\text{page\_part(relevant\_chunk(shift,chunk,(relevant\_single\_part(C\_spec))))}), \text{hp\_constr(C\_spec(content(content\_assignment(t1,e1), \ldots, content\_assignment(tn,\epsilon_n))))')' \)

then

\[
\llbracket T \rrbracket (P) = \text{C-ELEMENT}' = \text{C-ELEMENT} \cup \bigcup_{i=1}^{n} \{ \langle \text{newelement}_i, t_i, \epsilon_i \rangle \} \\
\text{C-SPEC}' = \text{C-SPEC} \cup \bigcup_{i=1}^{n} \{ \langle \text{c\_spec, newshift}_i, \text{newelement}_i \rangle \} \\
\exists \text{chunk}. \langle \text{chunk}, *, \text{c\_spec}, *, *, * \rangle \in \text{CHUNK} \land \\
\langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE} \} \\
\text{where } \forall \langle \text{c\_element}, *, * \rangle \in \text{C\_ELEMENT}. \\
\forall 1 \leq i \leq n. \text{newelement}_i \neq \text{c\_element} \land \\
\forall 1 \leq i, j \leq n. i = j \lor \text{newelement}_i \neq \text{newelement}_j \land \\
\forall \langle \text{c\_spec, shift}, * \rangle \in \text{C\_SPEC} \forall 1 \leq i \leq n. \text{newshift}_i \neq \text{shift} \land \\
\forall 1 \leq i, j \leq n. i = j \lor \text{newshift}_i \neq \text{newshift}_j
\]
if $T = \text{hp\_update(hp\_structure\_update(insert, (page\_part(relevant\_chunk(shift,chunk,(relevant\_single\_part(R\_spec)))), hp\_constr(R\_spec(rendering(rendering\_element(r_1),\ldots, rendering\_element(r_n))))))}$
then
$$[T](P) = \text{R\_ELEMENT'} = \text{R\_ELEMENT} \cup \bigcup_{i=1}^{n} \{\text{new\_element}_i, r_i\}$$
$$\text{R\_SPEC'} = \text{R\_SPEC} \cup \bigcup_{i=1}^{n} \{\langle r\_spec, \text{new\_shift}_i, \text{new\_element}_i \rangle\}$$
$$\exists \text{chunk}. \langle \text{chunk}, \ast, r\_spec, \ast, \ast \rangle \in \text{CHUNK} \land$$
$$\langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE}$$
where $\forall \langle \text{r\_element}, \ast \rangle \in \text{R\_ELEMENT}$.
$$\forall 1 \leq i \leq n. \text{new\_element}_i \neq \text{r\_element} \land$$
$$\forall 1 \leq i, j \leq n. i = j \lor \text{new\_element}_i \neq \text{new\_element}_j \land$$
$$\forall \langle r\_spec, \text{shift}, \ast \rangle \in \text{R\_SPEC}. \forall 1 \leq i \leq n. \text{new\_shift}_i \neq \text{shift} \land$$
$$\forall 1 \leq i, j \leq n. i = j \lor \text{new\_shift}_i \neq \text{new\_shift}_j$$

if $T = \text{hp\_update(hp\_structure\_update(insert, (page\_part(relevant\_chunk(shift_1,chunk,(relevant\_multi\_part(shift_2,exit\_point))))), hp\_constr(exit\_point(exit(uis))))})$)
then
$$[T](P) = \text{EXITPOINT'} = \text{EXITPOINT} \cup \{\langle \text{new\_exit}, u\_uis \rangle\}$$
$$\text{EXITPOINTSET'} = \text{EXITPOINTSET} \cup \{\langle \text{exit\_pointset}, \text{shift}_2, \text{new\_exit} \rangle\}$$
$$\exists \text{chunk}. \langle \text{chunk}, \ast, \ast, \ast, \ast, \text{exit\_pointset} \rangle \in \text{CHUNK} \land$$
$$\langle P, \text{chunk}, \text{shift}_1 \rangle \in \text{PAGE}$$
where $\forall \langle \text{exit\_point}, \ast \rangle \in \text{EXITPOINT}. \text{exit\_point} \neq \text{new\_exit}$
Meaning: Hyperpage Structure Update Insert Chunk

Given a `hp_structure_update` whose parameters are the token `insert`, a `page_part` whose parameters are a sequence of tokens denoting a specified chunk and a `hp_constr` whose parameter is a chunk as defined in the grammar for hyperpages, `hp_update` inserts that chunk into the page denoted by `P`.

if $T = 'hp_update(hp_structure_update(insert, (page_part(relevant_chunk(shift,chunk)), hp-constr(entry-point(entry(uis_1),...entry(uis_n))), (C-spec(content(content_assignment(t_1,e_1),...content_assignment(t_n,e_n))))), (R-spec(rendering(rendering_element(r_1),...rendering_element(r_n)))), (exit-point(exit(uis_1),...exit(uis_n))))'$

then
$$[[T]](P) = ENTROPYPOINT' = ENTROPYPOINT \cup \bigcup_{i=1}^{n} \{ \langle newentry_i, uis_i \rangle \}$$
$$ENTROPYPOINTSET' = ENTROPYPOINTSET \cup$$
$$\bigcup_{i=1}^{n} \{ \langle entrypointset, newshift_i, newentry_i \rangle \}$$
$$\exists chunk \langle chunk, entrypointset, *, *, * \rangle \in CHUNK \land$$
$$\exists \langle P, chunk, shift \rangle \in PAGE$$

where $\forall \langle entrypoint, * \rangle \in ENTROPYPOINT \land 1 \leq i \leq n.$

`newentry_i` $\neq$ `entrypoint` $\land$

$\forall 1 \leq i, j \leq n. \exists i, j \neq newentry_i \land newentry_j$ $\land$

$\forall \langle entrypointset, shift, * \rangle \in ENTROPYPOINTSET \land 1 \leq i \leq n.$

`newshift_i` $\neq$ `shift` $\land \forall 1 \leq i, j \leq n. \exists i, j \neq newshift_i \land newshift_j$

`C-ELEMENT' = C-ELEMENT $\cup \bigcup_{i=1}^{n} \{ \langle newelement_i, t_i, e_i \rangle \}$

`C-SPEC' = C-SPEC $\cup \bigcup_{i=1}^{n} \{ \langle c-spec, newshift_i, newelement_i \rangle \}$

$\exists chunk \langle chunk, *, c-spec, *, * \rangle \in CHUNK \land$

$\langle P, chunk, shift \rangle \in PAGE$

where $\forall \langle c-element, *, * \rangle \in C-ELEMENT.$

$\forall 1 \leq i \leq n. newelement_i \neq c-element \land$

$\forall 1 \leq i, j \leq n. i = j \lor newelement_i \neq newelement_j \land$

$\forall \langle c-spec, shift, * \rangle \in C-SPEC \land 1 \leq i \leq n. newshift_i \neq shift \land$

$\forall 1 \leq i, j \leq n. i = j \lor newshift_i \neq newshift_j$

`R-ELEMENT' = R-ELEMENT $\cup \bigcup_{i=1}^{n} \{ \langle newrelement_i, r_i \rangle \}$

`R-SPEC' = R-SPEC $\cup \bigcup_{i=1}^{n} \{ \langle r-spec, newshift_i, newelement_i \rangle \}$

$\exists chunk \langle chunk, *, r-spec, *, * \rangle \in CHUNK \land$

$\langle P, chunk, shift \rangle \in PAGE$

where $\forall \langle r-element, * \rangle \in R-ELEMENT.$

$\forall 1 \leq i \leq n. newrelement_i \neq r-element \land$

$\forall 1 \leq i, j \leq n. i = j \lor newelement_i \neq newelement_j \land$

$\forall \langle r-spec, shift, * \rangle \in R-SPEC \land 1 \leq i \leq n. newshift_i \neq shift \land$

$\forall 1 \leq i, j \leq n. i = j \lor newshift_i \neq newshift_j$
EXITPOINT' = EXITPOINT ∪ \bigcup_{i=1}^{n}\{(\text{newexit}_i, u_i)\}

EXITPOINTSET' = EXITPOINTSET ∪ \bigcup_{i=1}^{n}\{(\text{exitpointset}_i, \text{newshift}_i, \text{newexit}_i)\}

\exists\text{chunk}. \langle \text{chunk}, \ast, \ast, \ast, \ast, \text{exitpoint} \rangle \in \text{CHUNK} \land
\langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE}\}

where \forall\langle \text{exitpoint}, \ast \rangle \in \text{EXITPOINT}. \forall 1 \leq i \leq n. \text{newexit}_i \neq \text{exitpoint}_i \land
\forall 1 \leq i, j \leq n. i = j \lor \text{newexit}_i \neq \text{newexit}_j \land
\forall\langle \text{exitpointset}, \text{shift}, \ast \rangle \in \text{EXITPOINTSET}. \forall 1 \leq i \leq n. \text{newshift}_i \neq \text{shift}_i \land
\forall 1 \leq i, j \leq n. i = j \lor \text{newshift}_i \neq \text{newshift}_j

### Meaning: Hyperpage Structure Update Delete Page Part

Given a hp\_structure\_update whose parameters are the token delete and a page\_part whose parameters are a sequence of tokens denoting a specified page-part hp\_update deletes that page-part from a page in P.

if T = 'hp\_update(hp\_structure\_update(delete,
\text{page\_part(relevant\_chunk(shift, chunk))))'))'
then
$$\llbracket T \rrbracket_P = \text{CHUNK'} = \text{CHUNK} \setminus \{(\text{chunk}, \ast, \ast, \ast, \ast)\}$$
$$\langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE}\}

if T = 'hp\_update(hp\_structure\_update(delete,(page\_part(relevant\_chunk(shift_1, chunk,
(relevant\_single\_part(shift_2, entry-point)))))))'
then
$$\llbracket T \rrbracket_P = \text{ENTRYPOINT'} = \text{ENTRYPOINT} \setminus \{(\text{entrypoint}, \ast)\}$$
$$\exists\text{entrypointset}. \langle \text{entrypointset}, \text{shift}_2, \text{entrypoint} \rangle \in \text{ENTRYPOINTSET}$$
$$\wedge \exists\text{chunk}. \langle \text{chunk}, \text{entrypointset}, \ast, \ast, \ast \rangle \in \text{CHUNK} \land
\langle P, \text{chunk}, \text{shift}_1 \rangle \in \text{PAGE}\}

if T = 'hp\_update(hp\_structure\_update(delete,
\text{page\_part(relevant\_chunk(shift, chunk,(relevant\_single\_part(C-spec))))}))'
then
$$\llbracket T \rrbracket_P = \text{C-SPEC'} = \text{C-SPEC} \setminus \{(\text{c-spec}, \ast, \ast)\}$$
$$\exists\text{chunk}. \langle \text{chunk}, \ast, \text{c-spec}, \ast, \ast \rangle \in \text{CHUNK} \land
\langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE}\}

if T = 'hp\_update(hp\_structure\_update(delete,
\text{page\_part(relevant\_chunk(shift, chunk,(relevant\_single\_part(R-spec))))}))'
then
$$\llbracket T \rrbracket_P = \text{R-SPEC'} = \text{R-SPEC} \setminus \{(\text{r-spec}, \ast, \ast)\}$$
$$\exists\text{chunk}. \langle \text{chunk}, \ast, \ast, \text{r-spec}, \ast \rangle \in \text{CHUNK} \land
\langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE}\}
if \( T = 'hp\_update(hp\_structure\_update(delete, \(page\_part(relevant\_chunk(shift_1, chunk, relevant\_multi\_part(shift_2, exit-point))))))' \)

then
\[
[[T]](P) = EXITPOINT' = EXITPOINT \setminus \{\langle \text{exitpoint}, * \rangle\}
\exists \text{exitpointset}. \langle \text{exitpointset}, shift_2, \text{exitpoint} \rangle \in EXITPOINTSET \land \\
\exists \text{chunk}. \langle \text{chunk}, *, *, *, \text{exitpointset} \rangle \in \text{CHUNK} \land \\
\langle P, \text{chunk}, shift_1 \rangle \in \text{PAGE}\}
\]

Meaning : Hyperpage Structure Update Projection

Given a \( hp\_struture\_update \) whose parameters are a projection operator whose parameter is the token \( drop\_if \) (or the token \( retain\_if \)), and a selection condition on \( hp\_constr \), \( hp\_update \) deletes the page denoted by \( P \) from the relation \( \text{PAGE} \) if that page in the case of \( drop\_if \) meets the selection condition on \( hp\_constr \) and in the case of \( retain\_if \) does not meet that selection condition on \( hp\_constr \).

if \( T = 'hp\_update(hp\_structure\_update(projection\_operator(drop\_if)), selection\_condition\_on\_hp\_constr(\text{CHC}))' \)

then
\[
[[T]](P) = \text{PAGE}' = \text{PAGE} \setminus \{\langle P, *, * \rangle\}
\langle P, *, * \rangle \in \text{PAGE} \land [[\text{CHC}]](P)
\]

if \( T = 'hp\_update(hp\_structure\_update(projection\_operator(retain\_if)), selection\_condition\_on\_hp\_constr(\text{CHC}))' \)

then
\[
[[T]](P) = \text{PAGE}' = \text{PAGE} \setminus \{\langle P, *, * \rangle\}
\langle P, *, * \rangle \in \text{PAGE} \land \neg [[\text{CHC}]](P)
\]

Meaning : Selection Condition on Hyperpage Construct

Given an atomic selection condition on \( hp\_constr \), selection condition on \( hp\_constr \) tests a page denoted by \( P \) to see whether or not the atomic selection condition on \( hp\_constr \) is satisfied. A selection condition on \( hp\_constr \) may be proceeded by any of the tokens \( not \), \( and \), \( or \), and these are defined in the standard way.

if \( T = 'selection\_condition\_on\_hp\_constr(atomic\_selection\_condition\_on\_hp\_constr(\text{CHC}))' \)

then
\[
[[T]](P) = [[\text{CHC}]](P)
\]

if \( T = 'selection\_condition\_on\_hp\_constr(not, selection\_condition\_on\_hp\_constr(\text{CHC}))' \)
then
\[ T(P) = \neg \left[ \text{CHC}(P) \right] \]

if \( T = \text{'selection\_condition\_on\_hp\_constr(\text{and}, selection\_condition\_on\_hp\_constr(\text{CHC}_1), selection\_condition\_on\_hp\_constr(\text{CHC}_2)})' \)

then
\[ \left[ T \right](P) = \left[ \text{CHC}_1 \right](P) \land \left[ \text{CHC}_2 \right](P) \]

if \( T = \text{'selection\_condition\_on\_hp\_constr(\text{or}, selection\_condition\_on\_hp\_constr(\text{CHC}_1), selection\_condition\_no\_hp\_constr(\text{CHC}_2)})' \)

then
\[ \left[ T \right](P) = \left[ \text{CHC}_1 \right](P) \lor \left[ \text{CHC}_2 \right](P) \]

Meaning : Atomic Selection Condition on Hyperpage Construct

Given a sequence of values and tokens denoting a specified page-part i.e. \textit{shift}, \textit{chunk}, \textit{entry-point}, \textit{c-spec}, \textit{r-spec}, \textit{exit-point}, and a containment_expression whose parameters are the token \textit{contains}, and a regular expression, \textit{EXP}, \textit{atomic\_condition\_on\_hp\_constr} returns true (or false) if a page denoted by \( P \) has a specified page-part that contains the specified regular expression.

selects those pages in \( P \) whose specified page-part contains the specified regular expression.

if \( T = \text{'atomic\_condition\_on\_hp\_constr(page\_part((relevant\_chunk(shift, chunk))), containment\_expression(contains,EXP))'} \)

then
\[ \left[ T \right](P) = \exists \text{chunk}. \langle P, \text{shift}, \text{chunk} \rangle \in \text{PAGE} \land EXP \triangleleft \text{chunk} \]

if \( T = \text{'atomic\_condition\_on\_hp\_constr(page\_part(relevant\_chunk(shift_1,chunk,(relevant\_multi\_part(shift_2,entry-point)))), containment\_expression(contains,EXP))'} \)

then
\[ \left[ T \right](P) = \exists \text{chunk}. \langle P, \text{shift}_1, \text{chunk} \rangle \in \text{PAGE} \land \exists \text{entrypoint, entrypointset}. \langle \text{chunk, entrypointset, *, *, *} \rangle \in \text{CHUNK} \land \langle \text{entrypointset, shift}_2, \text{entrypoint} \rangle \in \text{ENTRYPOINTSET} \land EXP \triangleleft \text{entrypoint} \]

if \( T = \text{'atomic\_condition\_on\_hp\_constr(page\_part(relevant\_chunk(shift,chunk,(relevant\_single\_part(c-spec)))), containment\_expression(contains,EXP))'} \)

then
\[ \left[ T \right](P) = \exists \text{chunk}. \langle P, \text{shift}, \text{chunk} \rangle \in \text{PAGE} \land \exists \text{c-spec}. \langle \text{chunk, *, c-spec, *, *} \rangle \in \text{CHUNK} \land EXP \triangleleft \text{c-spec} \]
if \( T = ' \text{atomic\_condition\_on\_hp\_constr(page\_part(relevant\_chunk}
(shift,\text{chunk},(\text{relevant\_single\_part(R\_spec))))),
\text{containment\_expression(contains,}\text{EXP} ))' \)
then
\[
\mathcal{[[T]]}_\(P\) = \exists \text{chunk}. \langle P, shift, \text{chunk} \rangle \in \text{PAGE} \land
\exists \text{r\_spec}. \langle \text{chunk}, *,*,* \rangle \in \text{CHUNK} \land \text{EXP} \not\subset \text{r\_spec}
\]

if \( T = ' \text{atomic\_condition\_on\_hp\_constr(page\_part(relevant\_chunk}
(shift_1,\text{chunk},(\text{relevant\_multi\_part(shift_2,exit-point))))),
\text{containment\_expression(contains,}\text{EXP} ))' \)
then
\[
\mathcal{[[T]]}_\(P\) = \exists \text{chunk}. \langle P, shift_1, \text{chunk} \rangle \in \text{PAGE} \land
\exists \text{exitpoint, exitpointset}. \langle \text{chunk}, \text{exitpointset}, *,*,* \rangle \in \text{CHUNK} \land
\exists \text{exitpointset, shift}_2, \text{exitpoint} \rangle \in \text{EXITPOINTSET} \land \text{EXP} \not\subset \text{exitpoint}
\]

\textbf{Meaning : Hyperpage Terminal Rewrite}

Given a \text{hp\_terminal\_rewrite} whose parameters are a \text{rewrite} whose parameter is the token \text{page}, and a \text{from}, whose parameter is a regular expression denoted by \text{EXP}_1 and a \text{to} whose parameter is a regular expression denoted by \text{EXP}_2, \text{hp\_update} inserts a rewrite note in to the relation \text{NOTE} for a page denoted by \( P \). The note inserted specifies a rewriting action over the renderable text after it has been composed by the H-region, i.e., after content has been fetched and made ready for display (e.g., to map American into British spelling forms).

if \( T = ' \text{hp\_update(hp\_terminal\_rewrite(rewrite(scope(page),from(Exp}_1,to(Exp}_2))))' \)
then
\[
\mathcal{[[T]]}_\(P\) = \text{NOTE}'=\text{NOTE}\cup\{\langle \text{newnote}, P, \text{null, rewrite, null, page, Exp}_1, \text{Exp}_2, \text{null} \rangle \|
\langle \text{chunk}, *,*,*,* \rangle \in \text{CHUNK} \land
\langle P, \text{chunk}, * \rangle \in \text{PAGE} \}
\]
where \( \forall \langle \text{note}, *,*,*,*,*,*,* \rangle \in \text{NOTE} \text{.note} \neq \text{newnote} \)

\textbf{A.1.2 Extending the Meaning Function}

In the definitions for the Meaning function provided above a shift in the grammar for hyperpages and hyperpage annotations is assumed to represent a signed integer. The following subsection provides details of how to extend these definitions to include the case where a \text{shift} is the token \text{all}. Informally all that is required is to remove the constraint that a tuple refers to a specific part of a page at a specific shift (position). The effect of these is to refer to all parts rather than a specific subset.

For example the case
if $T = '\text{atomic\_condition(atomic\_condition\_on\_page\_part}$
\hspace{1em}page\_part((relevant\_chunk(shift, chunk)), \text{containment\_expression(contains, EXP))))'$
\hspace{1em}then
\hspace{1em}$[[T]] = \{\text{page}|(page, shift, chunk) \in \text{PAGE} \land EXP \triangleleft \text{chunk}\}$

would become

if $T = '\text{atomic\_condition(atomic\_condition\_on\_page\_part}$
\hspace{1em}page\_part((relevant\_chunk(all, chunk)), \text{containment\_expression(contains, EXP))))'$
\hspace{1em}then
\hspace{1em}$[[T]] = \{\text{page}|(page, *, *) \in \text{PAGE} \land EXP \triangleleft \text{chunk}\}$

and the case

if $T = '\text{atomic\_condition(atomic\_condition\_on\_page\_part}$
\hspace{1em}(page\_part(relevant\_chunk(shift1, chunk), (relevant\_single\_part(shift2, entry-point)))), \text{containment\_expression(contains, EXP))')'$
\hspace{1em}then
\hspace{1em}$[[T]] = \{\text{page}|(page, shift1, chunk) \in \text{PAGE} \land$
\hspace{2em}$\exists \text{entrypoint, entrypointset}. (\text{chunk, entrypointset, *, *, *}) \in \text{CHUNK} \land$
\hspace{3em}$\langle \text{entrypointset, shift2, entrypoint} \rangle \in \text{ENTRYPOINTSET} \land EXP \triangleleft \text{entrypoint}\}$

would become

if $T = '\text{atomic\_condition(atomic\_condition\_on\_page\_part}$
\hspace{1em}(page\_part(relevant\_chunk(all1, chunk), (relevant\_single\_part(all2, entry-point)))), \text{containment\_expression(contains, EXP))')'$
\hspace{1em}then
\hspace{1em}$[[T]] = \{\text{page}|(page, *, *) \in \text{PAGE} \land$
\hspace{2em}$\exists \text{entrypoint, entrypointset}. (\text{*, entrypointset, *, *, *}) \in \text{CHUNK} \land$
\hspace{3em}$\langle \text{entrypointset, *, *} \rangle \in \text{ENTRYPOINTSET} \land EXP \triangleleft \text{entrypoint}\}$

It is hoped that the above two examples illustrate the relative ease in which the case where
the shift in the grammar for hyperpages and hyperpage annotations is not a signed integer
but is the token all.

A.1.3 Functions for the Insertion and Deletion of Notes

The following section details two functions InsertNote and DeleteNote which are called
in the main body of the Meaning Function detailed in section .

Given an abstract syntax tree which denotes a specified page-part and a note, and a page
denoted by $P$, the function \textbf{InsertNote} detailed in figures A.1-A.3 inserts that \textit{note} into the relation \textit{NOTE} for the specified page-part.

Given an abstract syntax tree which denotes a specified page-part and a note, and a page denoted by $P$, the function \textbf{DeleteNote} detailed in figures A.4-A.6 deletes that \textit{note} from the relation \textit{NOTE} if that note is associated with the specified page-part.
```
InsertNote(`scope(page), attribute_assignment(attribute(a_1),attribute_value(v_1))
,...,attribute_assignment(attribute(a_n),attribute_value(v_n)))'(P)
    = NOTE'=NOTE∪\bigcup_{i=1}^{n}\{\langle newnote_i, P, null, attrib, page, null, a_i, v_i, null\rangle |
    \langle P, *, *\rangle ∈ PAGE\}
    \quad where ∀\langle note, *, *, *, *, *\rangle ∈ NOTE.∀1 ≤ i ≤ n.newnote_i \neq note ∧
    ∀1 ≤ i, j ≤ n.i = j \lor newnote_i \neq newnote_j

InsertNote(`scope(page_part(relevant_chunk(shift,chunk))),
attribute_assignment(attribute(a_1),attribute_value(v_1))
,...,attribute_assignment(attribute(a_n),attribute_value(v_n)))'(P)
    = NOTE'=NOTE∪\bigcup_{i=1}^{n}\{\langle newnote_i, P, chunk, attrib, chunk, null, a_i, v_i, null\rangle |
    \exists chunk.\langle chunk, *, *, *, *\rangle ∈ CHUNK ∧
    \langle P, chunk, shift\rangle ∈ PAGE\}
    \quad where ∀\langle note, *, *, *, *, *\rangle ∈ NOTE.∀1 ≤ i ≤ n.newnote_i \neq note ∧
    ∀1 ≤ i, j ≤ n.i = j \lor newnote_i \neq newnote_j

InsertNote(`scope(page_part(relevant_chunk(shift,chunk),
relevant_multi_part(shift_2.entry-point))),
attribute_assignment(attribute(a_1),attribute_value(v_1))
,...,attribute_assignment(attribute(a_n),attribute_value(v_n)))'(P)
    = NOTE'=NOTE∪\bigcup_{i=1}^{n}\{\langle newnote_i, P, chunk, attrib, entry, shift_2, a_i, v_i, null\rangle |
    \exists chunk.\langle chunk, *, *, *, *\rangle ∈ CHUNK ∧
    \langle P, chunk, shift\rangle ∈ PAGE\}
    \quad where ∀\langle note, *, *, *, *, *\rangle ∈ NOTE.∀1 ≤ i ≤ n.newnote_i \neq note ∧
    ∀1 ≤ i, j ≤ n.i = j \lor newnote_i \neq newnote_j

InsertNote(`scope(page_part(relevant_chunk(shift,chunk),
relevant_single_part(C-spec))),
attribute_assignment(attribute(a_1),attribute_value(v_1))
,...,attribute_assignment(attribute(a_n),attribute_value(v_n)))'(P)
    = NOTE'=NOTE∪\bigcup_{i=1}^{n}\{\langle newnote_i, P, chunk, attrib, cspec, null, a_i, v_i, null\rangle |
    \exists chunk.\langle chunk, *, *, *, *\rangle ∈ CHUNK ∧
    \langle P, chunk, shift\rangle ∈ PAGE\}
    \quad where ∀\langle note, *, *, *, *, *\rangle ∈ NOTE.∀1 ≤ i ≤ n.newnote_i \neq note ∧
    ∀1 ≤ i, j ≤ n.i = j \lor newnote_i \neq newnote_j

InsertNote(`scope(page_part(relevant_chunk(shift,chunk),
relevant_single_part(R-spec))),
attribute_assignment(attribute(a_1),attribute_value(v_1))
,...,attribute_assignment(attribute(a_n),attribute_value(v_n)))'(P)
    = NOTE'=NOTE∪\bigcup_{i=1}^{n}\{\langle newnote_i, P, chunk, attrib, rspec, null, a_i, v_i, null\rangle |
    \exists chunk.\langle chunk, *, *, *, *\rangle ∈ CHUNK ∧
    \langle P, chunk, shift\rangle ∈ PAGE\}
    \quad where ∀\langle note, *, *, *, *, *\rangle ∈ NOTE.∀1 ≤ i ≤ n.newnote_i \neq note ∧
    ∀1 ≤ i, j ≤ n.i = j \lor newnote_i \neq newnote_j

InsertNote(`scope(page_part(relevant_chunk(shift_1,chunk),
relevant_multi_part(shift_2.exit-point))),
attribute_assignment(attribute(a_1),attribute_value(v_1))
,...,attribute_assignment(attribute(a_n),attribute_value(v_n)))'(P)
    = NOTE'=NOTE∪\bigcup_{i=1}^{n}\{\langle newnote_i, P, chunk, attrib, exit, shift_2, a_i, v_i, null\rangle |
    \exists chunk.\langle chunk, *, *, *, *\rangle ∈ CHUNK ∧
    \langle P, chunk, shift\rangle ∈ PAGE\}
    \quad where ∀\langle note, *, *, *, *, *\rangle ∈ NOTE.∀1 ≤ i ≤ n.newnote_i \neq note ∧
    ∀1 ≤ i, j ≤ n.i = j \lor newnote_i \neq newnote_j
```

Figure A.1: Function : InsertNote(note) (I)
\begin{figure}[h]
\centering
\begin{align*}
\text{InsertNote} & (\text{rewrite(scope(page)}, \text{from}(EX P_1), \text{to}(EX P_2)) \rangle_p) \\
& = \text{NOTE}’ = \text{NOTE} U \{\langle \text{newnote}, P, \text{null}, \text{rewrite, page}, \text{null}, EX P_1, EX P_2, \text{null} \rangle\} \\
& \quad \langle P, *, * \rangle \in \text{PAGE} \\
& \quad \text{where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}. \text{note} \neq \text{newnote}
\end{align*}
\begin{align*}
\text{InsertNote} & (\text{rewrite(scope(page_part(relevant_chunk(shift, chunk)}), from}(EX P_1), \text{to}(EX P_2)) \rangle_p) \\
& = \text{NOTE}’ = \text{NOTE} U \{\langle \text{newnote}, P, \text{chunk}, \text{rewrite, chunk}, \text{null}, EX P_1, EX P_2, \text{null} \rangle\} \\
& \quad \exists \text{chunk}. \langle \text{chunk}, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE} \\
& \quad \text{where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}. \text{note} \neq \text{newnote}
\end{align*}
\begin{align*}
\text{InsertNote} & (\text{rewrite(scope(page_part(relevant_chunk(shift, chunk), relevant_multi_part(shift_2, entry_point)}), from}(EX P_1), \text{to}(EX P_2)) \rangle_p) \\
& = \text{NOTE}’ = \text{NOTE} U \{\langle \text{newnote}, P, \text{chunk}, \text{rewrite, entry, shift_2, EX P_1, EX P_2, null} \rangle\} \\
& \quad \exists \text{chunk}. \langle \text{chunk}, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE} \\
& \quad \text{where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}. \text{note} \neq \text{newnote}
\end{align*}
\begin{align*}
\text{InsertNote} & (\text{rewrite(scope(page_part(relevant_chunk(shift, chunk), relevant_single_part(C-spec)}), from}(EX P_1), \text{to}(EX P_2)) \rangle_p) \\
& = \text{NOTE}’ = \text{NOTE} U \{\langle \text{newnote}, P, \text{chunk}, \text{rewrite, cspec, null, EX P_1, EX P_2, null} \rangle\} \\
& \quad \exists \text{chunk}. \langle \text{chunk}, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE} \\
& \quad \text{where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}. \text{note} \neq \text{newnote}
\end{align*}
\begin{align*}
\text{InsertNote} & (\text{rewrite(scope(page_part(relevant_chunk(shift, chunk), relevant_single_part(R-spec)}), from}(EX P_1), \text{to}(EX P_2)) \rangle_p) \\
& = \text{NOTE}’ = \text{NOTE} U \{\langle \text{newnote}, P, \text{chunk}, \text{rewrite, rspec, null, EX P_1, EX P_2, null} \rangle\} \\
& \quad \exists \text{chunk}. \langle \text{chunk}, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE} \\
& \quad \text{where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}. \text{note} \neq \text{newnote}
\end{align*}
\begin{align*}
\text{InsertNote} & (\text{rewrite(scope(page_part(relevant_chunk(shift_1, chunk), relevant_multi_part(shift_2, exit-point)}), from}(EX P_1), \text{to}(EX P_2)) \rangle_p) \\
& = \text{NOTE}’ = \text{NOTE} U \{\langle \text{newnote}, P, \text{chunk}, \text{rewrite, exit, shift_2, EX P_1, EX P_2, null} \rangle\} \\
& \quad \exists \text{chunk}. \langle \text{chunk}, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, \text{shift_1} \rangle \in \text{PAGE} \\
& \quad \text{where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}. \text{note} \neq \text{newnote}
\end{align*}

Figure A.2: Function: \textbf{InsertNote(note)} (II)
\textbf{InsertNote} ('QOE, rewrite(scope(page), from(\textit{EXP}_1), to(\textit{EXP}_2))')(P) 
= NOTE'=NOTE \cup \{\langle \text{newnote}, P, null, rewrite, page, null, \text{EXP}_1, \text{EXP}_2, QOE \rangle | \\
\langle P, *, * \rangle \in \text{PAGE} \}
\text{ where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}.\text{note} \neq \text{newnote}
\textbf{InsertNote} ('QOE, rewrite(scope(page_part(relevant\_chunk(shift,chunk))), from(\textit{EXP}_1), to(\textit{EXP}_2))')(P) 
= NOTE'\cup\{\langle \text{newnote}, P, chunk, rewrite, chunk, null, \text{EXP}_1, \text{EXP}_2, QOE \rangle | \\
\exists \text{chunk}. \langle \text{chunk}, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, shift \rangle \in \text{PAGE} \}
\text{ where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}.\text{note} \neq \text{newnote}
\textbf{InsertNote} ('QOE, rewrite(scope(page_part(relevant\_chunk(shift,chunk), relevant\_multipart(shift_2,entry\_point))), from(\textit{EXP}_1), to(\textit{EXP}_2))')(P) 
= NOTE'\cup\{\langle \text{newnote}, P, \text{chunk}, rewrite, entry, shift_2, \text{EXP}_1, \text{EXP}_2, QOE \rangle | \\
\exists \text{chunk}. \langle \text{chunk}, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, shift \rangle \in \text{PAGE} \}
\text{ where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}.\text{note} \neq \text{newnote}
\textbf{InsertNote} ('QOE, rewrite(scope(page_part(relevant\_chunk(shift,chunk), relevant\_single\_part(C\_spec))), from(\textit{EXP}_1), to(\textit{EXP}_2))')(P) 
= NOTE'\cup\{\langle \text{newnote}, P, \text{chunk}, rewrite, cspec, null, \text{EXP}_1, \text{EXP}_2, QOE \rangle | \\
\exists \text{chunk}. \langle \text{chunk}, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, shift \rangle \in \text{PAGE} \}
\text{ where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}.\text{note} \neq \text{newnote}
\textbf{InsertNote} ('QOE, rewrite(scope(page_part(relevant\_chunk(shift,chunk), relevant\_single\_part(R\_spec))), from(\textit{EXP}_1), to(\textit{EXP}_2))')(P) 
= NOTE'\cup\{\langle \text{newnote}, P, \text{chunk}, rewrite, rspec, null, \text{EXP}_1, \text{EXP}_2, QOE \rangle | \\
\exists \text{chunk}. \langle \text{chunk}, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, shift \rangle \in \text{PAGE} \}
\text{ where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}.\text{note} \neq \text{newnote}
\textbf{InsertNote} ('QOE, rewrite(scope(page_part(relevant\_chunk(shift_1,chunk), relevant\_multipart(shift_2,exit\_point))), from(\textit{EXP}_1), to(\textit{EXP}_2))'(P) 
= NOTE'\cup\{\langle \text{newnote}, P, \text{chunk}, rewrite, exit, shift_2, \text{EXP}_1, \text{EXP}_2, QOE \rangle | \\
\exists \text{chunk}. \langle \text{chunk}, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, shift_1 \} \in \text{PAGE} \}
\text{ where } \forall \langle \text{note}, *, *, *, *, *, * \rangle \in \text{NOTE}.\text{note} \neq \text{newnote}

Figure A.3: Function: \textbf{InsertNote} (note) (III)
\[
\begin{align*}
\text{DeleteNote}(&\text{scope(page), attribute_assignment(attribute(a),attribute_value(v))}) \\
&\ldots, \text{attribute_assignment(attribute(a),attribute_value(v))})_{(P)} \\
&= \text{NOTE} = \text{NOTE} \setminus \bigcup_{i=1}^{n} \{(*, P, \text{chunk}, \text{attrib, page, null, } a_i, v_i, \text{null})\} \\
&\quad \langle P, *, * \rangle \in \text{PAGE} \\
\text{DeleteNote}(&\text{scope(page_part(relevant_chunk(shift,chunk))),} \\
&\quad \text{attribute_assignment(attribute(a),attribute_value(v))}) \\
&\ldots, \text{attribute_assignment(attribute(a),attribute_value(v))})_{(P)} \\
&= \text{NOTE} = \text{NOTE} \setminus \bigcup_{i=1}^{n} \{(*, P, \text{chunk}, \text{attrib, entry, shift, null, } a_i, v_i, \text{null})\} \\
&\quad \exists \text{chunk}. \langle \text{chunk}, *, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE} \\
\text{DeleteNote}(&\text{scope(page_part(relevant_chunk(shift,chunk),} \\
&\quad \text{relevant_multi_part(shift,entry-point))),} \\
&\quad \text{attribute_assignment(attribute(a),attribute_value(v))}) \\
&\ldots, \text{attribute_assignment(attribute(a),attribute_value(v))})_{(P)} \\
&= \text{NOTE} = \text{NOTE} \setminus \bigcup_{i=1}^{n} \{(*, P, \text{chunk}, \text{attrib, entry, shift, null, } a_i, v_i, \text{null})\} \\
&\quad \exists \text{chunk}. \langle \text{chunk}, *, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE} \\
\text{DeleteNote}(&\text{scope(page_part(relevant_chunk(shift,chunk),} \\
&\quad \text{relevant_single_part(C-spec))),} \\
&\quad \text{attribute_assignment(attribute(a),attribute_value(v))}) \\
&\ldots, \text{attribute_assignment(attribute(a),attribute_value(v))})_{(P)} \\
&= \text{NOTE} = \text{NOTE} \setminus \bigcup_{i=1}^{n} \{(*, P, \text{chunk}, \text{attrib, cspec, null, } a_i, v_i, \text{null})\} \\
&\quad \exists \text{chunk}. \langle \text{chunk}, *, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE} \\
\text{DeleteNote}(&\text{scope(page_part(relevant_chunk(shift,chunk),} \\
&\quad \text{relevant_single_part(R-spec))),} \\
&\quad \text{attribute_assignment(attribute(a),attribute_value(v))}) \\
&\ldots, \text{attribute_assignment(attribute(a),attribute_value(v))})_{(P)} \\
&= \text{NOTE} = \text{NOTE} \setminus \bigcup_{i=1}^{n} \{(*, P, \text{chunk}, \text{attrib, rspec, null, } a_i, v_i, \text{null})\} \\
&\quad \exists \text{chunk}. \langle \text{chunk}, *, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, \text{shift} \rangle \in \text{PAGE} \\
\text{DeleteNote}(&\text{scope(page_part(relevant_chunk(shift1,chunk),} \\
&\quad \text{relevant_multi_part(shift2,exit-point))),} \\
&\quad \text{attribute_assignment(attribute(a),attribute_value(v))}) \\
&\ldots, \text{attribute_assignment(attribute(a),attribute_value(v))})_{(P)} \\
&= \text{NOTE} = \text{NOTE} \setminus \bigcup_{i=1}^{n} \{(*, P, \text{chunk}, \text{attrib, exit, shift, null, } a_i, v_i, \text{null})\} \\
&\quad \exists \text{chunk}. \langle \text{chunk}, *, *, *, *, * \rangle \in \text{CHUNK} \land \langle P, \text{chunk}, \text{shift1} \rangle \in \text{PAGE} \\
\end{align*}
\]

Figure A.4: Function : \text{DeleteNote}(note) (I)
DeleteNote('\(\text{rewrite(scope(page), from}(\text{EXP}_1), \text{to}(\text{EXP}_2))\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{null, rewrite, page, null, \text{EXP}_1, \text{EXP}_2, \text{null}}, P, *, *) \notin \text{PAGE}\}\)

DeleteNote('\(\text{rewrite(scope(page, relevant\_chunk(shift, chunk)))}, \text{from}(\text{EXP}_1), \text{to}(\text{EXP}_2)\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{chunk, rewrite, chunk, null, \text{EXP}_1, \text{EXP}_2, \text{null}}), P, *, *, * \) \notin \text{CHUNK} \land (P, \text{chunk, shift}) \notin \text{PAGE}\}\)

DeleteNote('\(\text{rewrite(scope(page, relevant\_chunk(shift, chunk)), relevant\_multi\_part(shift_2, entry-point)))}, \text{from}(\text{EXP}_1), \text{to}(\text{EXP}_2)\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{chunk, rewrite, entry, shift_2, \text{EXP}_1, \text{EXP}_2, \text{null}}), P, *, *, * \) \notin \text{CHUNK} \land (P, \text{chunk, shift}) \notin \text{PAGE}\}\)

DeleteNote('\(\text{rewrite(scope(page, relevant\_chunk(shift, chunk), relevant\_single\_part(\text{c-spec}))})}, \text{from}(\text{EXP}_1), \text{to}(\text{EXP}_2)\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{chunk, rewrite, cspec, null, \text{EXP}_1, \text{EXP}_2, \text{null}}), P, *, *, * \) \notin \text{CHUNK} \land (P, \text{chunk, shift}) \notin \text{PAGE}\}\)

DeleteNote('\(\text{rewrite(scope(page, relevant\_chunk(shift, chunk), relevant\_single\_part(\text{r-spec}))})}, \text{from}(\text{EXP}_1), \text{to}(\text{EXP}_2)\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{chunk, rewrite, rspec, null, \text{EXP}_1, \text{EXP}_2, \text{null}}), P, *, *, * \) \notin \text{CHUNK} \land (P, \text{chunk, shift}) \notin \text{PAGE}\}\)

DeleteNote('\(\text{rewrite(scope(page, relevant\_chunk(shift_1, chunk), relevant\_multi\_part(shift_2, exit-point)))}, \text{from}(\text{EXP}_1), \text{to}(\text{EXP}_2)\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{chunk, rewrite, exit, shift_2, \text{EXP}_1, \text{EXP}_2, \text{null}}), P, *, *, * \) \notin \text{CHUNK} \land (P, \text{chunk, shift}) \notin \text{PAGE}\}\)

Figure A.5: Function : \textbf{DeleteNote}(\text{note}) (II)

DeleteNote('\(\text{\text{QOE, rewrite(scope(page), from}(\text{EXP}_1), \text{to}(\text{EXP}_2))}\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{null, rewrite, page, \text{null}, \text{EXP}_1, \text{EXP}_2, \text{QOE}}), P, *, *) \notin \text{PAGE}\}\)

DeleteNote('\(\text{\text{QOE, rewrite(scope(page, relevant\_chunk(shift, chunk)))}}, \text{from}(\text{EXP}_1), \text{to}(\text{EXP}_2)\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{chunk, rewrite, chunk, null, \text{EXP}_1, \text{EXP}_2, \text{QOE}}), P, *, *, * \) \notin \text{CHUNK} \land (P, \text{chunk, shift}) \notin \text{PAGE}\}\)

DeleteNote('\(\text{\text{QOE, rewrite(scope(page, relevant\_chunk(shift, chunk), relevant\_multi\_part(shift_2, entry-point)))}}, \text{from}(\text{EXP}_1), \text{to}(\text{EXP}_2)\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{chunk, rewrite, entry, shift_2, \text{EXP}_1, \text{EXP}_2, \text{QOE}}), P, *, *, * \) \notin \text{CHUNK} \land (P, \text{chunk, shift}) \notin \text{PAGE}\}\)

DeleteNote('\(\text{\text{QOE, rewrite(scope(page, relevant\_chunk(shift, chunk), relevant\_single\_part(\text{c-spec}))})}, \text{from}(\text{EXP}_1), \text{to}(\text{EXP}_2)\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{chunk, rewrite, cspec, \text{null}, \text{EXP}_1, \text{EXP}_2, \text{QOE}}), P, *, *, * \) \notin \text{CHUNK} \land (P, \text{chunk, shift}) \notin \text{PAGE}\}\)

DeleteNote('\(\text{\text{QOE, rewrite(scope(page, relevant\_chunk(shift, chunk), relevant\_single\_part(\text{r-spec}))})}, \text{from}(\text{EXP}_1), \text{to}(\text{EXP}_2)\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{chunk, rewrite, rspec, \text{null}, \text{EXP}_1, \text{EXP}_2, \text{QOE}}), P, *, *, * \) \notin \text{CHUNK} \land (P, \text{chunk, shift}) \notin \text{PAGE}\}\)

DeleteNote('\(\text{\text{QOE, rewrite(scope(page, relevant\_chunk(shift_1, chunk), relevant\_multi\_part(shift_2, exit-point)))}}, \text{from}(\text{EXP}_1), \text{to}(\text{EXP}_2)\)'\(^{(P)}\))
= NOTE'\(\text{NOTE} \setminus \{(*, P, \text{chunk, rewrite, exit, shift_2, \text{EXP}_1, \text{EXP}_2, \text{QOE}}), P, *, *, * \) \notin \text{CHUNK} \land (P, \text{chunk, shift}) \notin \text{PAGE}\}\)

Figure A.6: Function : \textbf{DeleteNote}(\text{note}) (III)
Appendix B
Procedural Design: The H-Region

Figures B.1 to B.9 detail the procedural design for the H-region. Functions for communication and interaction with servers and stores are not detailed. A standard parsing function appropriate for the grammar it is associated with is assumed and has been implemented (see Appendix D).
declare that
  request
    belongs_to {X|X is_recognized_by request_grammar}
  rendering_expression
    belongs_to {X|X is_a *string*} % might be parsed out with more structure
  outcome
    belongs_to {"succeeded","failed"}
  hyperpage
    belongs_to {X|X is_recognized_by hyperpage_grammar}
  hyperpage_program
    belongs_to {X|X is_parsed_out by hyperpage_grammar}
  hyperpage_reference
    belongs_to relative_hyperpage_reference U absolute_hyperpage_reference
  relative_hyperpage_reference
    belongs_to {"back","forward","fresh"}
  absolute_hyperpage_reference
    belongs_to local_hyperpage_reference U remote_hyperpage_reference
  local_hyperpage_reference
    belongs_to {X|X points_to_a_client_side_file}
  remote_hyperpage_reference
    belongs_to {X|X points_to_a_server_side_file}
  nature_of_hyperpage_reference
    belongs_to {"into_remote_server","into_log","into_hyperlibrary"}
  store
    belongs_to {"hyperlibrary","log","cache","local_memory"}
  log_position
    belongs_to {X|X is_a *pointer_into_tuple*} % implementation-specific
  key
    belongs_to local_hyperpage_reference U log_position
  column
    belongs_to { "hyperpage_id",
                "hyperpage",
                "rendering_expression",
                "outcome",
                "regular_expression",
                "from",
                "to" }
  value
    belongs_to local_hyperpage_reference
    U hyperpage
    U rendering_expression
    U outcome
    U {X|X is_a *regular_expression*}

Figure B.1: Procedural Design of the H-Region (1)
go_H-region(Log, Cache)
  <- R is_a request
  -> O is_a outcome
import
  append : (store,<column,value>, ..., <column,value>)
ask_current_UI_server_to_render : rendering_expression -> outcome
begin
  E is_a rendering_expression

  <O,E> := perform_retrieval(R)
  if O = "succeeded"
    then O := ask_current_UI_server_to_render(E)
  end_if
  if O = "succeeded"
    append("cache","hyperpage_id",R>
    ,<"rendering_expression",E>
    )
  end_if
  append("log","hyperpage_id",R>
    ,<"rendering_expression",E>
    ,<"outcome",O>
    )

  return(O)
end

Figure B.2: Procedural Design of the H-Region (2)

perform_retrieval
  <- R is_a hyperpage_reference
  -> <O is_a outcome, E is_a rendering_expression>
begin
  NHR is_a nature_of_hyperpage_reference

  NHR := classify_hyperpage_reference(R)

  case [NHR = "into_remote_server"]
    <O,E> := fetch_remote(R)
  case [NHR = "into_log"]
    <O,E> := fetch_logged(R)
  case [NHR = "into_hyperlibrary"]
    <O,E> := convert_to_rendering_expression(R)
  else <O,E> := "failed", ""
end_case

  return(<O,E>)
end

Figure B.3: Procedural Design of the H-Region (3)
classify_hyperpage_reference
  <- R is_a hyperpage_reference
  -> NHR is_a nature_of_hyperpage_reference
import
  find : (store,key)   -> outcome
  project : (store,key,column) -> value
begin
  0 is_a outcome

  if [R="back" or R="forward" or R="fresh"]
    then NHR := "into_log"
  else begin 0 := find("hyperlibrary",R)
    if [0 = "succeeded"]
      then NHR := "into_hyperlibrary"
    else NHR := "into_remote_server"
  end_if
end_if

return(NHR)
end

Figure B.4: Procedural Design of the H-Region (4)

fetch_remote
  <- R is_a hyperpage_reference
  -> <O is_a outcome, E is_a rendering_expression>
import
  ask_remote_HLBS_server : hyperpage_reference -> outcome X rendering_expression
begin
  <O,E> := ask_remote_HLBS_server(R)
  return(<O,E>)
end

Figure B.5: Procedural Design of the H-Region (5)
fetch_logged
  <- R is_a hyperpage_reference
  -> <O is_a outcome, E is_a rendering_expression>
note
  import
  find : (store,key) -> outcome
  project : (store,key,column) -> value

begin

/*---------------------------------------------------------------*/
/* CURRENT is global and is_a reference_into_log pointing to the record */
/* of the hyperpage which is currently visible to the user         */
/* POINT_TO is local                                                */
/* POINT_TO is_a reference_into_log                                */
/*---------------------------------------------------------------*/

POINT_TO := CURRENT

case [R="back" or R="forward"]
  if [R="back"]
    then POINT_TO := POINT_TO-1
    else POINT_TO := POINT_TO+1
  end_if
  O := find("log",POINT_TO)
  if [O = "succeeded"]
    then begin R := project("log","hyperpage_id",POINT_TO)
          E := project("cache","rendering_expression",R)
    end
    else E := ""
  end_if
end_case

CURRENT := POINT_TO
return(<O,E>)
end
convert_to_render
<- R is_a hyperpage_reference
-> <O is_a outcome, E is_a rendering_expression>
begin
HS is_a hyperpage_state
H is_a hyperpage
E,E' is_a rendering_expression

O := "failed"
E := ""

<O,H> := fetch_designed(R)
if [O = "succeeded"]
    then E := compose_hyperpage_program(parse_hyperpage(H))
end_if

return(<O,E>)
end

Figure B.7: Procedural Design of the H-Region (7)

fetch_designed
<- R is_a hyperpage_reference
-> <O is_a outcome, H is_a hyperpage>
import
find : (store,key) -> outcome
project : (store,key,column) -> value
begin
begin
    O := find("hyperlibrary",R)
if [O = "succeeded"]
    then H := project("hyperlibrary","hyperpage",R)
else H := ""
end_if
return(<O,H>)
end

Figure B.8: Procedural Design of the H-Region (8)
parse_hyperpage
<- H is_a hyperpage
-> P is_a hyperpage_program
begin
  % see hyperpage_grammar
end

compose_hyperpage_program
<- P is a hyperpage_program
-> E is_a rendering_expression
begin
  % see hyperpage abstract machine definition
end

Figure B.9: Procedural Design of the H-Region (9)
Appendix C

Procedural Design: The P-Region

Figure C.1 to C.16 detail the procedural design for the P-region. Functions for communication and interaction with servers and stores are not detailed. A standard parsing function appropriate for the grammar it is associated with is assumed and has been implemented (see Appendix D).
declare_that
request
  belongs_to {X|X is_a *string*} % the UIS might parse in more structure
rendering_expression
  belongs_to {X|X is_a *string*} % might be parsed out with more structure
outcome
  belongs_to {"succeeded", "failed"}
hyperpage
  belongs_to {X|X is_recognized_by hyperpage_grammar}
hyperpage_program
  belongs_to {X|X is_parsed_out by hyperpage_grammar}
hyperpage_reference
  belongs_to relative_hyperpage_reference U absolute_hyperpage_reference
relative_hyperpage_reference
  belongs_to {"back", "forward", "fresh"}
absolute_hyperpage_reference
  belongs_to local_hyperpage_reference U remote_hyperpage_reference
local_hyperpage_reference
  belongs_to {X|X points_to_a_client_side_file}
remote_hyperpage_reference
  belongs_to {X|X points_to_a_server_side_file}
annotation
  belongs_to {X|X is_recognized_by annotation_grammar}
nature_of_request
  belongs_to {"retrieve", "personalize","info_request"}
nature_of_hyperpage_reference
  belongs_to {"into_remote_server", "into_log", "into_hyperlibrary"}
hyperpage_state
  belongs_to {"designed", "personalized"}
personalization_request
  belongs_to {X|X is_recognized_by personalization_request_grammar}
personalization_program
  belongs_to {X|X is_parsed_out by personalization_request_grammar}
information_request
  belongs_to {X|X points_to_a_client_side_file}
store
  belongs_to {"hyperlibrary", "log", "cache", "local_memory"}
log_position
  belongs_to {X|X is_a *pointer_into_tuple*} % implementation-specific
key
  belongs_to local_hyperpage_reference U log_position
column
  belongs_to {"hyperpage_id", "hyperpage", "annotation", "rendering_expression", "regular_expression", "from", "to"}
value
  belongs local_hyperpage_reference U hyperpage U annotation U rendering_expression U {X|X is_a *regular_expression*}

Figure C.1: Basic Design of the P-region (1)
Go_P-region(Log, Cache)
<- R is_a request
-> O is_a outcome
import
append : (store, <column, value>, ..., <column, value>)
begin
NR is_a nature_of_request
E is_a rendering_expression

NR := classify_request(R)

case [NR = "retrieve"]
  <O, E> := perform_retrieval(R)
  append("cache", <"rendering_expression", E>)
  append("log", <"hyperpage_id", R>)
  case [NR = "personalize"]
    O := perform_personalization(R)
  case [NR = "info_request"]
    O := perform_annotation_retrieval(R)
  else O := "failed"
end_case

return(O)
end

Figure C.2: Basic Design of the P-region (2)

classify_request
<- R is_a request
-> NR is_a nature_of_request
begin
  case [R is_a hyperpage_reference]
    NR := "retrieve"
  case [R is_a personalization_request]
    NR := "personalize"
  case [R is_a information_request]
    NR := "info_request"
  else NR := ""
end_case

return(NR)
end

Figure C.3: Basic Design of the P-region (3)
perform_retrieval
<- R is_a hyperpage_reference
-> <O is_a outcome, E is_a rendering_expression>
begin
NHR is_a nature_of_hyperpage_reference

NHR := classify_hyperpage_reference(R)

case [NHR = "into_remote_server"]
 O, E := fetch_remote(R)
case [NHR = "into_log"]
 O, E := fetch_logged(R)
case [NHR = "into_hyperlibrary"]
 O, E := convert_to_rendering_expression(R)
else <O, E> := <"failed", "">
end_case

return(<O, E>)
end

Figure C.4: Basic Design of the P-region (4)

classify_hyperpage_reference
<- R is_a hyperpage_reference
-> NHR is_a nature_of_hyperpage_reference
import
 find : (store, key)  -> outcome
project : (store, key, column)  -> value
begin
 O is_a outcome

if [R="back" or R="forward" or R="fresh"]
 then NHR := "into_log"
else begin 0 := find("hyperlibrary", R)
 if [O = "succeeded"]
 then NHR := "into_hyperlibrary"
 else NHR := "into_remote_server"
 end_if
end_if

return(NHR)
end

Figure C.5: Basic Design of the P-region (5)
fetch_remote
<- R is_a hyperpage_reference
-> <O is_a outcome, E is_a rendering_expression>
import
ask_remote_HLBS_server: hyperpage_reference -> outcome X rendering_expression
begin
    <O, E> := ask_remote_HLBS_server(R)
    return(<O, E>)
end

Figure C.6: Basic Design of the P-region (6)

fetch_logged
<- R is_a hyperpage_reference
-> <O is_a outcome, E is_a rendering_expression>
import
    find : (store, key) -> outcome
    project : (store, key, column) -> value
begin
    % CURRENT is global and points to current log record
    POINT_TO is local and points to some log record

    POINT_TO := CURRENT
    case [R="back" or R="forward"]
        if [R="back"]
            then POINT_TO := POINT_TO-1
        else POINT_TO := POINT_TO+1
        end_if
    O := find("log", POINT_TO)
    if [O = "succeeded"]
        then begin R := project("log", "hyperpage_id", POINT_TO)
            E := project("cache", "rendering_expression", R)
        end
    else E := ""
    end_if
    case [R="fresh"]
        O := find("log", POINT_TO)
        if [O = "succeeded"]
            then begin R := project("log", "hyperpage_id", POINT_TO)
                <O, E> := convert_to_rendering_expression(R)
            end
        else E := ""
        end_if
    end_case
    return(<O, E>)
end

Figure C.7: Basic Design of the P-region (7)
convert_to_render
<- R is_a hyperpage_reference
-> <O is_a outcome, E is_a rendering_expression>
begin
HS is_a hyperpage_state
H is_a hyperpage
A is_a annotation
E, E’ is_a rendering_expression

HS := classify_hyperpage_state(R)

case [HS = "designed"]
  <O, H> := fetch_designed(R)
  if [O = "succeeded"]
    then E := compose_hyperpage_program(parse_hyperpage(H))
    else E := ""
  end_if
end_case

case [HS = "personalized"]
  <O, H, A> := fetch_personalized(R)
  if [O = "succeeded"]
    then begin E’ := compose_hyperpage_program(parse_hyperpage(H))
          E := enforce_preferences(A, E’)
    end
    else E := ""
  else begin O := "failed"
           E := ""
    end
  end_case

return(<O, E>)
end

Figure C.8: Basic Design of the P-region (8)
classify_hyperpage_state
<- R is_a hyperpage_reference
-> NS is_a hyperpage_state
import
find : (store, key) -> outcome
project : (store, key, column) -> value
begin
O is_a outcome
O := find("hyperlibrary", R)
if [O = "succeeded"]
then begin A := project("hyperlibrary", "annotation", R)
if [A = "]"
then NS := "designed"
else NS := "personalized"
end_if
end
else NS := ""
end_if
return(NS)
end

Figure C.9: Basic Design of the P-region (9)

fetch_designed
<- R is_a hyperpage_reference
-> <O is_a outcome, H is_a hyperpage>
import
find : (store, key) -> outcome
project : (store, key, column) -> value
begin
O := find("hyperlibrary", R)
if [O = "succeeded"]
then H := project("hyperlibrary", "hyperpage", R)
else H := ""
end_if
return(<O, H>)
end

Figure C.10: Basic Design of the P-region (10)
fetch_personalized
<- R is_a hyperpage_reference
-> <O is_a outcome, H is_a hyperpage, A is_a annotation>

import
find : (store, key) -> outcome
project : (store, key, column) -> value

begin
O := find("hyperlibrary", R)
if [O = "succeeded"]
then begin H := project("hyperlibrary", "hyperpage", R)
     A := project("hyperlibrary", "annotation", R)
end
else begin H := ""
     A := ""
end
end_if
return(<O, H, A>)


Figure C.11: Basic Design of the P-region (11)

apply_preferences
<- <A is_a annotation, E' is_a rendering_expression>
-> E is_a rendering_expression

import
project : (store, key, column) -> value
replace_regexp : (regexp_with__markers, regexp_with__markers) -> value

begin
N is_a note
FROM is_a regexp_with__markers % as in Emacs
TO is_a regexp_with__markers % as in Emacs

for_each N in A
   if [N is_a rewrite]
      then begin FROM := project("local_memory", "from", N)
              TO := project("local_memory", "to", N)
              E := replace_regexp(FROM, TO, E') % as in Emacs
         end
   else E := E'
end_if
end_for
return(E)

end


Figure C.12: Basic Design of the P-region (12)
parse_hyperpage
<- H is_a hyperpage
-> P is_a hyperpage_program
begin
  % see grammar
end

compose_hyperpage_program
<- P is_a hyperpage_program
-> E is_a rendering_expression
begin
  % see hyperpage abstract machine definition
end

Figure C.13: Basic Design of the P-region (13)

perform_personalization
<- R is_a personalization_request
-> O is_a outcome
begin
  O := interpret_personalization_program(parse_personalization_request(R))
  return(O)
end

Figure C.14: Basic Design of the P-region (14)

parse_personalization_request
<- R is_a personalization_request
-> P is_a personalization_program, O is_a outcome
begin
  O := parse_personalization_request(R)
  % see grammar for personalization requests
end

interpret_personalization_program
<- P is_a personalization_program, O is_a outcome
-> E is_a rendering_expression
begin
  case [O = "succeeded"]
    O := interpret_personalization_program(P)
  else O := "failed"
  end_case

  % see personalization_request formal semantics
  return(O, E)
end

Figure C.15: Basic Design of the P-region (15)
perform_annotation_retrieval(R)
<- R is_a hyperpage_reference
-> <O is_a outcome, A is_a annotation>
import
find : (store, key) -> outcome
project : (store, key, column) -> value
begin
O := find("hyperlibrary", R)
if [O = "succeeded"]
then begin
    A := project("hyperlibrary", "annotation", R)
end
else begin H := ""
    A := ""
end
end_if
return(<O, A>)
end

Figure C.16: Basic Design of the P-region (16)
Appendix D
A Prolog Parser for Hyperpages, Hyperpage Annotations and Personalisation Requests

This section reproduces the prolog program that implements the back-end of PAS. Only the core files used are reproduced here. These core files make calls to support files used for “pretty printing” and reading and writing to text files. Support files are not reproduced here but are available upon request.

D.1 Parser defining DCG grammars for personalisation requests, hyperpages and hyperpage annotations

This section details the specification of a DCG prolog program used to implement a parser for personalisation requests, hyperpages and hyperpage annotations. This parser performs lexical and syntactic analysis for personalisation requests, hyperpages and hyperpage annotations and performs semantic analysis of personalisation requests.

% Adaptive, Personalizable Hyperlink-Based System
%
% James Ohene-Djan

% File: aphlbs_grammars.pl
% By: James Ohene-Djan
% When: Tue Nov 24 10:24:30 GMT 1998

% What: defines DCG grammars for personalization requests, annotations and hyperpage specs

% Note: assumes token sequence has been generated by

253
% Richard O'Keefe's tokenizer rdtok.pl
% Personalization Requests

personalization_request(personalization_request(AS,AL)) -->
  action_scope(AS),
  actionlist(AL).

action_scope(action_scope(SPF,C)) -->
  t_select_page_if(SPF),
  condition(C).

actionlist('..'(A)) -->
  action(A).

actionlist('..'(A,AL)) -->
  action(A),
  actionlist(AL).

action(action(ATD,AU)) -->
  t_an_then_do(ATD),
  t_lbrace(_TLB),
  an_update(AU),
  t_rbrace(_TRB).

action(action(HTD,HU)) -->
  t_hp_then_do(HTD),
  t_lbrace(_TLB),
  hp_update(HU),
  t_rbrace(_TRB).

an_update(annotation_update(UO,A)) -->
  update_operator(UO),
  annotation(A).

an_update(annotation_update(UO,PC,N)) -->
  update_operator(UO),
  persistency_condition(PC),
  notelist(N).

an_update(annotation_update(UO,N)) -->
  update_operator(UO),
  notelist(N).

an_update(annotation_update(NP)) -->
  note_projection(NP).

an_update(annotation_update(AASO)) -->
  attribute_assignment_set_operation(AASO).
attribute_assignment_set_operation(attribute_assignment_set_operation(PP1,SO,PP2)) -->
    page_part(PP1),
    set_operator(SO),
    page_part(PP2).

set_operator(set_operation(TU)) -->
    t_union(TU).
set_operator(set_operation(TI)) -->
    t_intersection(TI).

update_operator(update_operation(TI)) -->
    t_insert(TI).
    update_operator(update_operation(TD)) -->
    t_delete(TD).

persistency_condition(persistency_condition(TU,QE)) -->
    t_until(TU),
    query_on_environment(QE).
persistency_condition(persistency_condition(TW),QE) -->
    t_while(TW),
    query_on_environment(QE).

condition(condition(AC)) -->
    atomic_condition(AC).
condition(condition(TUO,C)) -->
    unary_boolean_operator(TUO),
    condition(C).
condition(condition(C1,TBO,C2)) -->
    atomic_condition(C1),
    binary_boolean_operator(TBO),
    condition(C2).

atomic_condition(atomic_condition(ACP)) -->
    atomic_condition_on_page(ACP).
atomic_condition(atomic_condition(ACPP)) -->
    atomic_condition_on_page_part(ACPP).
atomic_condition(atomic_condition(ACN)) -->
    atomic_condition_on_note(ACN).
atomic_condition(atomic_condition(TT)) -->
    t_true(TT).
atomic_condition_on_page(atomic_condition_on_page(TP,CE)) -->
    t_page(TP),
    containment_expression(CE).

atomic_condition_on_page_part(atomic_condition_on_page_PP,CE)) -->
    page_part(PP),
    containment_expression(CE).

note_projection(note_projection(PO,CN)) -->
    projection_operator(PO),
    condition_on_note(CN).

projection_operator(projection_operation(TDI)) -->
    t_drop_if(TDI).

projection_operator(projection_operation(TRI)) -->
    t_retain_if(TRI).

condition_on_note(condition_on_note(ACN)) -->
    atomic_condition_on_note(ACN).

condition_on_note(condition_on_note(TUO,CN)) -->
    unary_boolean_operator(TUO),
    condition_on_note(CN).

condition_on_note(condition_on_note(CN,TBO,CN)) -->
    condition_on_note(CN),
    binary_boolean_operator(TBO),
    condition_on_note(CN).

atomic_condition_on_note(atomic_condition_on_note(RN,CE)) -->
    relevant_note(RN),
    containment_expression(CE).

relevant_note(relevant_note(S,TN)) -->
    shift(S),
    t_comma(_TC),
    t_note(TN).

hp_update(hp_update(HSU)) -->
    hp_structure_update(HSU).

hp_update(hp_update(HTR)) -->
    hp_terminal_rewrite(HTR).

hp_structure_update(hp_structure_update(TD,PP)) -->
    t_delete(TD),
page_part(PP).
hp_structure_update(hp_structure_update(TI,PP,HC)) -->
t_insert(TI),
page_part(PP),
hp_constr(HC).
hp_structure_update(hp_structure_update(PO,CHC)) -->
projection_operator(PO),
condition_on_hp_constr(CHC).
hp_constr(hp_construct(HP)) →
    hyperpage(HP).
hp_constr(hp_construct(C)) →
    chunk(C).
hp_constr(hp_construct(EnP)) →
    entry_point_element(EnP).
hp_constr(hp_construct(CS)) →
    c_spec(CS).
hp_constr(hp_construct(RS)) →
    r_spec(RS).
hp_constr(hp_construct(ExP)) →
    exit_point_element(ExP).

condition_on_hp_constr(condition_on_hp_construct(ACHC)) →
    atomic_condition_on_hp_constr(ACHC).
condition_on_hp_constr(condition_on_hp_construct(TUO,CHC)) →
    unary_boolean_operator(TUO),
    condition_on_hp_constr(CHC).
condition_on_hp_constr(condition_on_hp_construct(CHC1,TBO,CHC2)) →
    condition_on_hp_constr(CHC1),
    binary_boolean_operator(TBO),
    condition_on_hp_constr(CHC2).

atomic_condition_on_hp_constr(atomic_condition_on_hp_construct(PP,CE)) →
    page_part(PP),
    containment_expression(CE).

hp_terminal_rewrite(hp_terminal_rewrite(R)) →
    rewrite(R).

containment_expression(containment_expression(TC,RE)) →
    t_contains(TC),
    regular_expression(RE).

% Annotations

annotation(annotation(TA,NL)) →
    t.annotation(TA),
    t.lbrace(_TLB),
    notelist(NL),
    t.rbrace(_TRB).
notelist(N) -->
  note(N),
  t_semicolon(_TSC).

note(note(SC,AAL)) -->
  scope(SC),
  attribute_assignment_list(AAL).

note(note(R)) -->
  rewrite(R).

note(note(QE,R)) -->
  query_on_environment(QE),
  rewrite(R).

scope(scope(TP)) -->
  t_page(TP),
  t_colon(_TC).

scope(scope(PP)) -->
  page_part(PP),
  t_colon(_TC).

page_part(page_part(RC)) -->
  t_lsquare(_TLS),
  relevant_chunk(RC),
  t_rsquare(_TRS).

page_part(page_part(RC,RSP)) -->
  t_lsquare(_TLS),
  relevant_chunk(RC),
  t_lround(_TLR),
  relevant_single_part(RSP),
  t_rround(_TRR),
  t_rsquare(_TRS).

page_part(page_part(RC,RMP)) -->
  t_lsquare(_TLS),
  relevant_chunk(RC),
  t_lround(_TLR),
  relevant_multi_part(RMP),
  t_rround(_TRR),
  t_rsquare(_TRS).
relevant_chunk(relevant_chunk(S,TC)) -->
    shift(S),
    t_comma(_TC),
    t_chunk(TC).

relevant_single_part(relevant_single_part(TCS)) -->
    t_c_spec(TCS).
relevant_single_part(relevant_single_part(TRS)) -->
    t_r_spec(TRS).

relevant_multi_part(relevant_multi_part(S,TEXP)) -->
    shift(S),
    t_comma(_TC),
    t_entry_point(TEXP).
relevant_multi_part(relevant_multi_part(S,TEP)) -->
    shift(S),
    t_comma(_TC),
    t_exit_point(TEP).

shift(shift(TA)) -->
    t_all(TA).
shift(shift(TS)) -->
    t_some(TS).
shift(shift(SI)) -->
    signed_integer(SI).

signed_integer(signed_integer(‘+’,I)) -->
    int(I).
signed_integer(signed_integer(TS,I)) -->
    t_sign(TS),
    int(I).

attribute_assignment_list(AA) -->
    attribute_assignment(AA).

attribute_assignment(attribute_assignment(A,V)) -->
    attribute(A),
    t_assignment(_TA),
    attribute_value(V).
attribute_value(V) -->
 value(V).
attribute_value(EnPL) -->
 entry_point(EnPL).
attribute_value(ExPL) -->
 exit_point(ExPL).

rewrite(rewrite(S,F,T)) -->
 scope(S),
 from(F),
 t_arrow(_TA),
 to(T).

from(from(RE)) -->
 regular_expression(RE).

to(to(RE)) -->
 regular_expression(RE).

% Hyperpage Specifications

hyperpage(hyperpage(TP,CL)) -->
 t_page(TP),
 t_lbrace(_TLB),
 chunk_list(CL),
 t_rbrace(_TRB).

chunk_list('..'(C)) -->
 chunk(C).
chunk_list('..'(C,CL)) -->
 chunk(C),
 chunk_list(CL).
chunk(chunk(EnPL,C_spec,R_spec,ExPL)) -->
  t_chunk(_TC),
  t_lbrace(_TLB),
  entry_point(EnPL),
  c_spec(C_spec),
  r_spec(R_spec),
  exit_point(ExPL),
  t_rbrace(_TRB).

chunk(chunk(EnPL,C_spec,R_spec)) -->
  t_chunk(_TC),
  t_lbrace(_TLB),
  entry_point(EnPL),
  c_spec(C_spec),
  r_spec(R_spec),
  t_rbrace(_TRB).

chunk(chunk(C_spec,R_spec,ExPL)) -->
  t_chunk(_TC),
  t_lbrace(_TLB),
  c_spec(C_spec),
  r_spec(R_spec),
  exit_point(ExPL),
  t_rbrace(_TRB).

chunk(chunk(C_spec,R_spec)) -->
  t_chunk(_TC),
  t_lbrace(_TLB),
  c_spec(C_spec),
  r_spec(R_spec),
  t_rbrace(_TRB).

entry_point(entry_point(EnPE)) -->
  t_entry_point(_TEP),
  t_lbrace(_TLB),
  entry_point_contents(EnPE),
  t_rbrace(_TRB).

entry_point_contents('..'(EnP)) -->
  entry_point_element(EnP).

entry_point_contents('..'(EnP,EnPE)) -->
  entry_point_element(EnP),
  entry_point_contents(EnPE).
entry_point_element(epel(UIS)) →
   uis_string(UIS).

c_spec(c_spec(CAL)) →
   t_c_spec(_TCS),
   t_lbrace(_TLB),
   content_assignment_list(CAL),
   t_rbrace(_TRB).

content_assignment_list('..'(CA)) →
   content_assignment(CA).
content_assignment_list('..'(CA,CAL)) →
   content_assignment(CA),
   t_comma(_TC),
   content_assignment_list(CAL).

content_assignment(content_assignment(TV,V)) →
   template_variable(TV),
   t_assignment(_TA),
   value(V).
content_assignment(content_assignment(TV,DBS)) →
   template_variable(TV),
   t_assignment(_TA),
   t_lsquare(_TLS),
   dbs_string(DBS),
   t_rsquare(_TRS).

r_spec(r_spec(RTL)) →
   t_r_spec(_TRS),
   t_lbrace(_TLB),
   rendering_template(RTL),
   t_rbrace(_TRB).

rendering_template('..'(RT)) →
   rendering_template_element(RT).
rendering_template('..'(RT,RTL)) →
   rendering_template_element(RT),
   rendering_template(RTL).

rendering_template_element(rel(UIS)) →
   uis_string(UIS).
rendering_template_element(rel(TV)) -->
    template_variable(TV).

exit_point(exit_point(ExPE)) -->
    t_exit_point(_TEXP),
    t_lbrace(_TLB),
    exit_point_contents(ExPE),
    t_rbrace(_TRB).

exit_point_contents('..'(EXnP)) -->
    exit_point_element(EXnP).
exit_point_contents('..'(EXnP,ExPE)) -->
    exit_point_element(EXnP),
    exit_point_contents(ExPE).

exit_point_element(exel(UIS)) -->
    uis_string(UIS).

% LEXICON: terminal classes and tokens

% Terminal Classes

query_on_environment(query_on_environment(query_on_environment(query_on_environment))) -->
    [query_on_environment]. % has to be properly done
query_on_environment(query_on_environment(query_on_environment(query_on_environment))) -->
    []. % has to be properly done

% Lexical Categories

% NOTE: assumes Richard O'Keefe's tokenizer

attribute('DESCRIPTION') --> [atom(description)].
attribute('KEYWORD') --> [atom(keyword)].
attribute('LEVEL') --> [atom(level)].
attribute('SEE_AS_WELL') --> [atom(see_as_well)].
attribute('SEE_INSTEAD') --> [atom(see_instead)].
attribute('WHEREFROM') --> [atom(wherefrom)].
attribute('WHERETO') --> [atom(whereto)].
attribute('STATUS') --> [atom(status)].

int(I) --> [integer(I)].
value(I) --> [integer(I)].
value(A) --> [atom(A)].
value(S) --> [string(S)].

regular_expression(RE) --> [atom(RE)].
uis_string(UIS) --> [string(UIS)].
dbs_string(DBS) --> [string(DBS)].
template_variable(TV) --> [atom(TV)].
binary_boolean_operator('AND') --> [atom(and)].
binary_boolean_operator('OR') --> [atom(or)].
unary_boolean_operator('NOT') --> [atom(not)].

% Tokens

t_until('UNTIL') --> [atom(until)].
t_while('WHILE') --> [atom(while)].
t_an_then_do('ANNOTATION_THEN_DO') --> [atom(an_then_do)].
t_hp_then_do('HP_THEN_DO') --> [atom(hp_then_do)].
t_select_page_if('SELECT_PAGE_IF') --> [atom(select_page_if)].
t_union('UNION') --> [atom(union)].
t_intersection('INTERSECTION') --> [atom(intersection)].
t_true('TRUE') --> [atom(true)].
t_note('NOTE') --> [atom(note)].
t_page('PAGE') --> [atom(page)].
t_chunk('CHUNK') --> [atom(chunk)].
t_c_spec('CONTENT') --> [atom(content)].
t_r_spec('RENDERING') --> [atom(rendering)].
t_entry_point('ENTRY') --> [atom(entry)].
t_exit_point('EXIT') --> [atom(exit)].
t_all('ALL') --> [atom(all)].
t_some('SOME') --> [atom(some)].
t_sign('+') --> [atom('+')].
t_sign('-') --> [atom('-')].
t_drop_if('DROP_IF') --> [atom(drop_if)].
t_retain_if('RETAIN_IF') --> [atom(retain_if)].
t_annotation('ANNOTATION') --> [atom(annotation)].
t_arrow('->') --> [atom('->')].
t_colon('::') --> [atom('::')].
t_semicolon('::') --> [atom('::')].
t_assignment('::=') --> [atom('::=')].
t_contains('CONTAINS') --> [atom(contains)].
t_insert('INSERT') --> [atom(insert)].
t_delete('DELETE') --> [atom(delete)].
t_lbrace('{') --> ['{'].
t_rbrace('}') --> ['}'].
t_lround('(') --> ['(']. % note the extra space
t_rround(')') --> [')'].
t_rsquare(']') --> [']'].
t_lsquare('[') --> ['['].
t_comma(',') --> [','].
D.2 Loader for lexical, syntactic and semantic analysers in PAS

This section details the prolog code used to assign the meaning to personalisation requests, thereby generating their corresponding SQL representation.

This file makes calls to several utility functions not reproduced here by are available upon request.

% Adaptive, Personalizable Hyperlink-Based System
% % James Ohene-Djan
% File: aphlbs_reader.pl
% When: Tue Nov 24 10:24:30 GMT 1998
% aphlbs_reconsult/2 - this is like reconsult, ought to be able to
% call this from the toplevel.

aphlbs_reconsult(What, File, Return) :-
    abolish(What/2),
    seeing(OldInput),
    see(File), % Open file File
    repeat,
    aphlbs_read(What, batch, T), % Read/Parse a What into term T
    aphlbs_reconsult_assert(T), % Process it
    T == end_of_file, % Loop back if not at end of file
    !,
    seen, % Close the file
    see(OldInput),
    Return = done.

aphlbs_reconsult_assert(end_of_file) :- !.
aphlbs_reconsult_assert(T) :- assertz(T).

aphlbs_read(What, Mode, Answer) :-
    read_tokens(Tokens, _Vars),
    (Tokens = [] -> Answer = end_of_file
 ; (( Phrase =.. [What, A, Tokens, []], call(Phrase))
    -> Answer = A,
    nl, write(What),
    write(‘ parsing succeeded’), nl,
aphlbs_feedback(Answer, Mode, _Reply),
aphlbs_semantics(Answer, SQL),
write(SQL)

; nl, write(What),
write(' parsing failed with this token sequence:'),
nl, nl, write(Tokens), nl, fail
).

aphlbs_semantics(personalization_request(AS, AL), SQL) :-
aphlbs_semantics(AS, SQL1), write(SQL1),
aphlbs_semantics(AL, SQL2),
concat_atom([SQL1, '

', SQL2], SQL).
% Newlines between selection and action

aphlbs_semantics(action_scope('SELECT_PAGE_IF', C), SQL) :-
aphlbs_semantics(C, SQL1),
concat_atom(['INSERT PAGE.page, PAGE.chunk, PAGE.shift INTO SELECTEDPAGES
', ' WHERE PAGE.page IN (SELECT PAGE.page FROM PAGE)
', ' AND (PAGE.page IN 
', SQL1, ')
', ')
', SQL]).

aphlbs_semantics(condition(atomic_condition('TRUE')), '(select page from PAGE)')

aphlbs_semantics(condition(atomic_condition(atomic_condition_on_page(
    page_part(relevant_chunk(shift(SC1), 'CHUNK'),
    relevant_multi_part(shift(signed_integer('+',I)), 'EXIT')),
    containment_expression('CONTAINS', S))), SQL) :-
% S is the string to match against, which is concat'ed in below
concat_atom(['(SELECT PAGE.page FROM PAGE 
', ' WHERE PAGE.chunk IN 
', ' (SELECT CHUNK.chunk FROM CHUNK 
', ' WHERE CHUNK.entrypointset IN 
', ' (SELECT ENTRYPOINTSET.entrypointset FROM ENTRYPOINTSET 
', ' WHERE ENTRYPOINTSET.shift = "',I,'" ) 
', ' AND ENTRYPOINTSET.exitpoint IN 
', ' (SELECT ENTRYPOINT.exitpoint FROM ENTRYPOINT 
', ' WHERE ENTRYPOINT.x_string LIKE "', S, '")) 
', ' AND (PAGE.shift = "',SC1,'" ) 
', ')
', SQL).  % concat all of above, put result in SQL

aphlbs_semantics(condition(atomic_condition(atomic_condition_on_page
(page_part(relevant_chunk(shift(SC1), 'CHUNK'),
  relevant_single_part('CONTENT'),
  containment_expression('CONTAINS', S))),SQL) :-

% S is the string to match against, which is concat'ed in below
concat_atom(['(SELECT PAGE.page FROM PAGE 

  WHERE PAGE.chunk IN \n',
  ' (SELECT CHUNK.chunk FROM CHUNK \n',
  ' WHERE CHUNK.cspec IN \n',
  ' (SELECT CSPEC.cspec FROM CSPEC \n',
  ' WHERE CSPEC.c-element IN \n',
  ' (SELECT C-ELEMENT.c-element FROM C-ELEMENT \n',
  ' WHERE C-ELEMENT.c_string LIKE "', S, '") ) \n',
  ' AND (PAGE.shift = "',SC1,'") \n']

  SQL).  % concat all of above, put result in SQL

aphlbs_semantics(condition(atomic_condition(atomic_condition_on_page
  (page_part(relevant_chunk(shift(SC1), 'CHUNK'),
    relevant_single_part('RENDERING'),
    containment_expression('CONTAINS', S))),SQL) :-

% S is the string to match against, which is concat'ed in below
concat_atom(['(SELECT PAGE.page FROM PAGE 

  WHERE PAGE.chunk IN \n',
  ' (SELECT CHUNK.chunk FROM CHUNK \n',
  ' WHERE CHUNK.rspec IN \n',
  ' (SELECT RSPEC.cspec FROM RSPEC \n',
  ' WHERE RSPEC.r-element IN \n',
  ' (SELECT R-ELEMENT.r-element FROM R-ELEMENT \n',
  ' WHERE R-ELEMENT.r_string LIKE "', S, '") ) \n',
  ' AND (PAGE.shift = "',SC1,'") \n']

  SQL).  % concat all of above, put result in SQL

aphlbs_semantics(condition(atomic_condition(atomic_condition_on_page
  (page_part(relevant_chunk(shift(SC1), 'CHUNK'),
    relevant_multi_part(shift(signed_integer('+',I)), 'EXIT'),
    containment_expression('CONTAINS', S))),SQL) :-

% S is the string to match against, which is concat'ed in below
concat_atom(['(SELECT PAGE.page FROM PAGE 

  WHERE PAGE.chunk IN \n',
  ' (SELECT CHUNK.exitpointset IN \n',
  ' (SELECT EXITPOINTSET.exitpointset FROM EXITPOINTSET \n',
  ' WHERE EXITPOINTSET.shift = "',I,'") \n']

  SQL).  % concat all of above, put result in SQL

269
AND EXITPOINTSET.exitpoint IN \n',
(SELECT EXITPOINT.exitpoint FROM EXITPOINT \n',
WHERE EXITPOINT.x_string LIKE "' S, ''")\n',
AND (PAGE.shift = "' SC1,"\n', SQL). % concat all of above, put result in SQL
aphlbs_semantics(condition(atomic_condition(atomic_condition_on_note
(relevant_note(shift(signed_integer('+', SC1)), 'NOTE'),
containment_expression('CONTAINS', S))),SQL) :-

% S is the string to match against, which is concat'ed in below
concat_atom(['(SELECT PAGE.page FROM PAGE 
', '
WHERE PAGE.chunk IN 
', '
(SELECT CHUNK.chunk FROM CHUNK 
', '
WHERE CHUNK.chunk IN 
', '
(SELECT NOTE.chunk FROM NOTE 
', '
WHERE NOTE.n_string LIKE ",', S, "])) 
', '
AND (PAGE.shift = ",', SC1, ") 
', SQL). % concat all of above, put result in SQL
aphlbs_semantics(condition(atomic_condition(atomic_condition_on_page
  (page_part(relevant_chunk(shift(SC1), 'CHUNK')),
  containment_expression('CONTAINS', S))),SQL) :-
concat_atom(['(SELECT PAGE.page FROM PAGE 
', 
  ' AND PAGE.chunk IN \n', 
  ' (SELECT CHUNK.chunk FROM CHUNK \n', 
  ' WHERE CHUNK.entrypointset IN\n', 
  ' (SELECT ENTRYPOINTSET.entrypointset FROM ENTRYPOINTSET\n', 
  ' WHERE ENTRYPOINTSET.entrypoint IN\n', 
  ' (SELECT ENTRYPOINT.entrypoint FROM ENTRYPOINT\n', 
  ' WHERE ENTRYPOINT.e_string LIKE ",S,")))\n', 
  '\n', 
  ' OR PAGE.page IN\n', 
  ' (SELECT PAGE.page FROM PAGE \n', 
  ' WHERE PAGE.chunk IN \n', 
  ' (SELECT CHUNK.chunk FROM CHUNK \n', 
  ' WHERE CHUNK.cspec IN\n', 
  ' (SELECT CSPEC.cspec FROM CSPEC \n', 
  ' WHERE CSPEC.c-element IN\n', 
  ' (SELECT C-ELEMENT.c-element FROM C-ELEMENT\n', 
  ' WHERE C-ELEMENT.c_string LIKE ",S,")))\n', 
  '\n', 
  ' OR PAGE.page IN\n', 
  ' (SELECT PAGE.page FROM PAGE \n', 
  ' WHERE PAGE.chunk IN \n', 
  ' (SELECT CHUNK.chunk FROM CHUNK \n', 
  ' WHERE CHUNK.rspec IN\n', 
  ' (SELECT RSPEC.rspec FROM RSPEC \n', 
  ' WHERE RSPEC.r-element IN\n', 
  ' (SELECT R-ELEMENT.r-element FROM R-ELEMENT\n', 
  ' WHERE R-ELEMENT.r_string LIKE ",S,")))\n', 
  '\n', 
  ' OR PAGE.page IN\n', 
  ' (SELECT PAGE.page FROM PAGE \n', 
  ' WHERE PAGE.chunk IN \n', 
  ' (SELECT CHUNK.chunk FROM CHUNK \n', 
  ' WHERE CHUNK.exitpointset IN\n', 
  ' (SELECT EXITPOINTSET.entrypointset FROM EXITPOINTSET \n', 
  ' WHERE EXITPOINTSET.entrypoint IN\n', 
  ' (SELECT EXITPOINT.exitpoint FROM EXITPOINT \n', 
  ' WHERE EXITPOINT.x_string LIKE ",S,"))) \n', 
  ' AND (PAGE.shift = ",SC1,"\n'), SQL].

272
aphlbs_semantics(condition(atomic_condition(atomic_condition_on_page("PAGE", containment_expression('CONTAINS', S))), SQL) :-
    concat_atom(['(SELECT PAGE.page FROM PAGE \n', 'AND PAGE.chunk IN \n', '  (SELECT CHUNK.chunk FROM CHUNK \n', '  WHERE CHUNK.entrypointset IN\n', '  (SELECT ENTRYPPOINTSET.entrypointset FROM ENTRYPPOINTSET\n', '  WHERE ENTRYPPOINTSET.entrypoint IN\n', '  (SELECT ENTRYPPOINT.entrypoint FROM ENTRYPPOINT\n', '  WHERE ENTRYPPOINT.e_string LIKE ",", S, ",")\n', ', \n', '  OR PAGE.page IN\n', '  (SELECT PAGE.page FROM PAGE \n', '  WHERE PAGE.chunk IN \n', '  (SELECT CHUNK.chunk FROM CHUNK \n', '  WHERE CHUNK.cspec IN\n', '  (SELECT CSPEC.cspec FROM CSPEC \n', '  WHERE CSPEC.c-element IN\n', '  (SELECT C-ELEMENT.c-element FROM C-ELEMENT\n', '  WHERE C-ELEMENT.c_string LIKE ",", S, ",")\n', ', \n', '  OR PAGE.page IN\n', '  (SELECT PAGE.page FROM PAGE \n', '  WHERE PAGE.chunk IN \n', '  (SELECT CHUNK.chunk FROM CHUNK \n', '  WHERE CHUNK.rspec IN\n', '  (SELECT RSPEC.rspec FROM RSPEC \n', '  WHERE RSPEC.r-element IN\n', '  (SELECT R-ELEMENT.r-element FROM R-ELEMENT\n', '  WHERE R-ELEMENT.r_string LIKE "", S, ",")\n', ', \n', '  OR PAGE.page IN\n', '  (SELECT PAGE.page FROM PAGE \n', '  WHERE PAGE.chunk IN \n', '  (SELECT CHUNK.chunk FROM CHUNK \n', '  WHERE CHUNK.exitpointset IN\n', '  (SELECT EXITPOINTSET.entrypointset FROM EXITPOINTSET \n', '  WHERE EXITPOINTSET.exitpoint IN\n', '  (SELECT EXITPOINT.exitpoint FROM EXITPOINT\n', '  WHERE EXITPOINT.x_string LIKE "", S, ",")\n', ')]), SQL).
aphlbs_semantics(A, SQL) :-
aphlbs_semantics(A, SQL).

aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update
   ('DELETE', page_part(relevant_chunk(shift(signed_integer(+,I)),'CHUNK')))), SQL) :-
concat_atom(['(DELETE * FROM PAGE 
   , WHERE PAGE.page IN 
   , (SELECT page FROM SELECTEDPAGES) 
   , AND (PAGE.shift = "",I,"")]
   SQL).

aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update
   (projection_operation('RETAIN_IF'), condition_on_hp_construct
   (atomic_condition_on_hp_construct(page_part(relevant_chunk
   (shift(signed_integer(+, SC1)), 'CHUNK')),
   containment_expression('CONTAINS', S))))), SQL) :-
% S is the string to match against, which is concat'ed in below
concat_atom(['DELETE * FROM PAGE 
   , WHERE PAGE.page IN 
   , (SELECT page FROM SELECTEDPAGES) 
   , AND PAGE.chunk NOT IN 
   , (SELECT CHUNK.chunk FROM CHUNK 
   , WHERE CHUNK.entrypointset IN
   , (SELECT ENTRYPONTSET.entrypointset FROM ENTRYPONTSET
   , WHERE ENTRYPONTSET.entrypoint IN
   , (SELECT ENTRYPONT.entréepoint FROM ENTRYPONT
   , WHERE ENTRYPONT.e_string LIKE "", S, ")")
   AND (PAGE.shift = ",SC1,"
   
   OR PAGE.page IN 
   , (SELECT page FROM SELECTEDPAGES) 
   , AND PAGE.chunk NOT IN 
   , (SELECT CHUNK.chunk FROM CHUNK 
   , WHERE CHUNK.cspec IN
   , (SELECT CSPEC.cspec FROM CSPEC 
   , WHERE CSPEC.c-element IN
   , (SELECT C-ELEMENT.c-element FROM C-ELEMENT
   , WHERE C-ELEMENT.c_string LIKE ", S, ")")])
   , 274
aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update
('DELETE', page_part(relevant_chunk(shift(signed_integer('+', SC1)), 'CHUNK'),
relevent_multi_part(shift(signed_integer('+',I)), 'ENTRY'))))), SQL) :-
concat_atom(['(DELETE * FROM ENTRYPOINT 
', 
WHERE (ENTRYPOINT.shift ="',I,'" ) \n', 
AND ENTRYPOINT.entrypoint IN \n', 
(SELECT ENTRYPOINTSET.entrypoint FROM ENTRYPOINTSET \n', 
WHERE ENTRYPOINTSET.entrypointset IN \n', 
(SELECT CHUNK.entrypointset FROM CHUNK \n', 
WHERE CHUNK.chunk IN \n', 
(SELECT SELECTEDPAGES.chunk FROM SELECTEDPAGES))) \n', 
AND (PAGE.shift = "',SC1,'")\n'],
SQL).
aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update
('DELETE', page_part(relevant_chunk(shift(signed_integer('+', SC1)), 'CHUNK'),
relevant_single_part('CSPEC')))), SQL) :-
concat_atom(['(DELETE * FROM CSPEC 
', ' WHERE CSPEC.cspec IN 
', ' (SELECT CHUNK.cspec FROM CHUNK 
', ' WHERE CHUNK.chunk IN 
', ' (SELECT SELECTEDPAGES.chunk FROM SELECTEDPAGES)))) 
', ' AND (PAGE.shift = ",',SC1,'"))\\n'], SQL).

aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update
('DELETE', page_part(relevant_chunk(shift(signed_integer('+', SC1)), 'CHUNK'),
relevant_single_part('RSPEC')))), SQL) :-
concat_atom(['(DELETE * FROM RSPEC 
', ' WHERE RSPEC.rspec IN 
', ' (SELECT CHUNK.rspec FROM CHUNK 
', ' WHERE CHUNK.chunk IN 
', ' (SELECT SELECTEDPAGES.chunk FROM SELECTEDPAGES)))) 
', ' AND (PAGE.shift = ",',SC1,'"))\\n'], SQL).

aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update
('DELETE', page_part(relevant_chunk(shift(signed_integer('+', SC1)), 'CHUNK'),
relevant_multi_part(shift(signed_integer('+', I)), 'EXIT')))), SQL) :-
concat_atom(['(DELETE * FROM EXITPOINT 
', ' WHERE (EXITPOINT.shift ="',I,'")) 
', ' AND (EXITPOINT.exitpoint IN 
', ' (SELECT EXITPOINTSET.exitpoint FROM EXITPOINTSET 
', ' WHERE EXITPOINTSET.exitpointset IN 
', ' (SELECT CHUNK.exitpointset FROM CHUNK 
', ' WHERE CHUNK.chunk IN 
', ' (SELECT SELECTEDPAGES.chunk FROM SELECTEDPAGES)))) 
', ' AND (PAGE.shift = ",',SC1,'"))\\n'], SQL).
aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update(projection_operation('DROP_IF'), condition_on_hp_construct(atomic_condition_on_hp_construct(page_part(relevant_chunk(shift(signed_integer('+', SC1)), 'CHUNK'), relevant_multi_part(shift(signed_integer('+', I)), 'ENTRY')), containment_expression('CONTAINS', S))))), SQL) :-
concat_atom(['(DELETE * FROM PAGE 
', ' WHERE PAGE.page IN 
', ' (SELECT page FROM SELECTEDPAGES) 
', ' AND PAGE.chunk IN 
', ' (SELECT CHUNK.chunk FROM CHUNK 
', ' WHERE CHUNK.entrypointset IN 
', ' (SELECT ENTRYPOINTSET.entrypointset FROM ENTRYPOINTSET 
', ' WHERE ENTRYPOINTSET.entrypoint IN 
', ' (SELECT ENTRYPOINT.entrypoint FROM ENTRYPOINT 
', ' WHERE ENTRYPOINT.e_string LIKE "', S, '")) 
', ' AND (ENTRYPOINT.shift = "', I, '" ) 
', ' AND (PAGE.shift = "', SC1, '" )']
, SQL).

aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update(projection_operation('DROP_IF'), condition_on_hp_construct(atomic_condition_on_hp_construct(page_part(relevant_chunk(shift(signed_integer('+', SC1)), 'CHUNK'), relevant_single_part('CONTENT')), containment_expression('CONTAINS', S))))), SQL) :-
concat_atom(['(DELETE * FROM PAGE 
', ' WHERE PAGE.page IN 
', ' (SELECT page FROM SELECTEDPAGES) 
', ' AND PAGE.chunk IN 
', ' (SELECT CHUNK.chunk FROM CHUNK 
', ' WHERE CHUNK.cspec IN 
', ' (SELECT CSPEC.cspec FROM CSPEC 
', ' WHERE CSPEC.c-element IN 
', ' (SELECT C-ELEMENT.c-element FROM C-ELEMENT 
', ' WHERE C-ELEMENT.c_string LIKE "', S, '")) 
', ' AND (C-ELEMENT.shift = "', I, '" ) 
', ' AND (PAGE.shift = "', SC1, '" ))']
, SQL). % concat all of above, put result in SQL

277
aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update
(projection_operation('DROP_IF'), condition_on_hp_construct
(atomic_condition_on_hp_construct(page_part(relevant_chunk
(shift(signed_integer('+', SC1)), 'CHUNK'),
re relevant_single_part('RENDERING'),
containment_expression('CONTAINS', S))))), SQL) :-
concat_atom(['(DELETE * FROM PAGE \n,
  WHERE PAGE.page IN \n,
  (SELECT page FROM SELECTEDPAGES) \n,
  AND PAGE.chunk IN \n,
  (SELECT CHUNK.chunk FROM CHUNK \n,
  WHERE CHUNK.rspec IN \n,
  (SELECT RSPEC.rspec FROM RSPEC \n,
  WHERE RSPEC.r-element IN \n,
  (SELECT R-ELEMENT.r-element FROM R-ELEMENT \n,
  WHERE R-ELEMENT.r_string LIKE ",S,"))) \n,
  AND (PAGE.shift = ",SC1," ) \n], SQL). % concat all of above, put result in SQL

aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update
(projection_operation('DROP_IF'), condition_on_hp_construct
(atomic_condition_on_hp_construct(page_part(relevant_chunk
(shift(signed_integer('+', SC1)), 'CHUNK'),
relevant_multi_part(shift(signed_integer('+', I)), 'EXIT'),
containment_expression('CONTAINS', S))))), SQL) :-
concat_atom(['(DELETE * FROM PAGE \n,
  WHERE PAGE.page IN \n,
  (SELECT page FROM SELECTEDPAGES) \n,
  AND PAGE.chunk IN \n,
  (SELECT CHUNK.chunk FROM CHUNK \n,
  WHERE CHUNK.entrypointset IN \n,
  (SELECT EXITPOINTSET.entrypointset FROM EXITPOINTSET \n,
  WHERE EXITPOINTSET.exitpoint IN \n,
  (SELECT EXITPOINT.exitpoint FROM EXITPOINT \n,
  WHERE EXITPOINT.x_string LIKE ",S,"))) \n,
  AND (EXITPOINT.shift =",I," ) \n,
  AND (PAGE.shift =",SC1," )\n], SQL).

aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update
(projection_operation('RETAIN_IF'), condition_on_hp_construct
...
(atomic_condition_on_hp_construct(page_part(relevant_chunk
(shift(signed_integer('+', SC1)), 'CHUNK'),
relevant_multi_part(shift(signed_integer('+', I)), 'ENTRY')),
containment_expression('CONTAINS', S))))), SQL) :-
concat_atom(['(DELETE * FROM PAGE 
',
WHERE PAGE.page IN 
',
(SELECT page FROM SELECTEDPAGES) 
',
AND PAGE.chunk NOT IN 
',
(SELECT CHUNK.chunk FROM CHUNK 
',
WHERE CHUNK.entrypointset IN 
',
(SELECT ENTRYPONSET.entrypointset FROM ENTRYPONSET 
',
WHERE ENTRYPONSET.entrypoint IN 
',
(SELECT ENTRYPONSET.entrypoint FROM ENTRYPONSET 
',
WHERE ENTRYPONSET.e_string LIKE "', S, '"')) 
',
AND (ENTRYPOINT.shift ="',I,'" ) 
',
AND (PAGE.shift ="',SC1,'" ))
'], SQL).
aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update
(projection_operation('RETAIN_IF'), condition_on_hp_construct
(atomic_condition_on_hp_construct(page_part(relevant_chunk
(shift(signed_integer('+', SC1)), 'CHUNK'),
relevant_single_part('RENDERING'),
containment_expression('CONTAINS', S)))), SQL) :-
concat_atom(['(DELETE * FROM PAGE 
', 'WHERE PAGE.page IN 
', 'AND PAGE.chunk NOT IN 
', 'WHERE CHUNK.cspec IN 
', 'WHERE CSPEC.c-element IN 
', 'WHERE R-SPEC.rspec IN 
', 'WHERE R-ELEMENT.r-element IN 
', 'WHERE R-ELEMENT.r_string LIKE 
', 'AND (PAGE.shift = 
', 'SQL). % concat all of above, put result in SQL

aphlbs_semantics(action('HP_THEN_DO', hp_update(hp_structure_update
(projection_operation('RETAIN_IF'), condition_on_hp_construct
(atomic_condition_on_hp_construct(page_part(relevant_chunk
(shift(signed_integer('+', SC1)), 'CHUNK'),
relevant_single_part('CONTENT'),
containment_expression('CONTAINS', S)))), SQL) :-
concat_atom(['(DELETE * FROM PAGE 
', 'WHERE PAGE.page IN 
', 'AND PAGE.chunk NOT IN 
', 'WHERE CHUNK.cspec IN 
', 'WHERE CSPEC.cspec FROM CSPEC 
', 'WHERE CHUNK.cspec IN 
', 'WHERE CHUNK.cspec IN 
', 'WHERE CSPEC.cspec FROM CSPEC 
', 'WHERE CSPEC.cspec FROM CSPEC 
', 'WHERE RSPEC.rspec FROM RSPEC 
', 'WHERE R-ELEMENT.r-element IN 
', 'WHERE R-ELEMENT.r_element FROM R-ELEMENT 
', 'WHERE R-ELEMENT.r_string LIKE 
', 'AND (PAGE.shift = 
', 'SQL). % concat all of above, put result in SQL

280
aphlbs_semantics(action(’HP_THEN_DO’, hp_update(hp_structure_update
(projection_operation(’RETAIN_IF’), condition_on_hp_construct
(atomic_condition_on_hp_construct(page_part(relevant_chunk
(shift(signed_integer(’+’, SC1)), ’CHUNK’),
relevant_multi_part(shift(signed_integer(’+’, I)), ’EXIT’)),
containment_expression(’CONTAINS’, S)))))), SQL) :concat_atom([’(DELETE * FROM PAGE \n’,
’

WHERE PAGE.page IN \n’,

’

(SELECT page FROM SELECTEDPAGES) \n’,

’

AND PAGE.chunk NOT IN \n’,

’

(SELECT CHUNK.chunk FROM CHUNK \n’,

’

WHERE CHUNK.entrypointset IN \n’,

’

(SELECT EXITPOINTSET.entrypointset FROM

’

WHERE EXITPOINTSET.exitpoint IN \n’,

’

(SELECT EXITPOINT.exitpoint FROM EXITPOINT \n’,

’

WHERE EXITPOINT.x_string LIKE "’, S, ’")))) \n’,

’

AND (EXITPOINT.shift ="’,I,’" )

’

AND (PAGE.shift ="’,SC1,’" ))\n’],

EXITPOINTSET \n’,

\n’,

SQL).
aphlbs_semantics(action(’HP_THEN_DO’, hp_update(hp_structure_update
(projection_operation(’DROP_IF’), condition_on_hp_construct
(atomic_condition_on_hp_construct(page_part(relevant_chunk
(shift(signed_integer(+, SC1)), ’CHUNK’)),
containment_expression(’CONTAINS’, S)))))), SQL) :% S is the string to match against, which is concat’ed in below
concat_atom([’DELETE * FROM PAGE

\n’,

’

WHERE PAGE.page IN \n’,

’

(SELECT page FROM SELECTEDPAGES) \n’,

’

AND PAGE.chunk IN \n’,

’

(SELECT CHUNK.chunk FROM CHUNK \n’,

’

WHERE CHUNK.entrypointset IN\n’,

’

(SELECT ENTRYPOINTSET.entrypointset FROM

’

WHERE ENTRYPOINTSET.entrypoint IN\n’,

’

(SELECT ENTRYPOINT.entrypoint FROM ENTRYPOINT\n’,

’

WHERE ENTRYPOINT.e_string LIKE "’, S, ’"))))\n’,

’

AND (PAGE.shift = "’,SC1,’")\n’,

’

\n’,

’

OR

’

(SELECT page FROM SELECTEDPAGES) \n’,

’

AND

’

(SELECT CHUNK.chunk FROM CHUNK \n’,

PAGE.page IN \n’,
PAGE.chunk IN \n’,

281

ENTRYPOINTSET\n’,


aphlbs_semantics(action('ANNOTATION_THEN_DO', annotation_update
  (update_operation('DELETE'), annotation('ANNOTATION',
    note(scope('PAGE'), attribute_assignment(A, V))))), SQL) :-
concat_atom(['(DELETE * FROM NOTE 
  WHERE NOTE.page IN 
  (SELECT page FROM SELECTEDPAGES) 
  AND (NOTE.scope = "page")
  AND (PAGE.lhs = ",A,"))
  AND (PAGE.rhs = ",V,"))
, SQL).
aphlbs_semantics(action('ANNOTATION_THEN_DO', annotation_update
(update_operation('DELETE'), annotation('ANNOTATION',
note(scope(page_part(relevant_chunk
(shift(signed_integer('+', I)), 'CHUNK'))),
attribute_assignment(A, V))))), SQL) :-
concat_atom(['(DELETE * FROM NOTE
', ' WHERE NOTE.page IN
', ' (SELECT page FROM SELECTEDPAGES
', ' WHERE SELECTEDPAGES.shift ="'+I,'"
', ' AND (NOTE.scope = "chunk")
', ' AND (PAGE.lhs = "'+A,'"
', ' AND (PAGE.rhs = "'+V,'")
']), SQL).

aphlbs_semantics(action('ANNOTATION_THEN_DO', annotation_update
(update_operation('DELETE'), annotation('ANNOTATION',
note(scope(page_part(relevant_chunk
(shift(signed_integer('+', I)), 'CHUNK'))),
relevance_multi_part(shift(signed_integer('+', SC1)), 'ENTRY'))),
attribute_assignment(A, V))))), SQL) :-
concat_atom(['(DELETE * FROM NOTE
', ' WHERE NOTE.page IN
', ' (SELECT page FROM SELECTEDPAGES)
', ' WHERE SELECTEDPAGES.shift ="'+I,'"
', ' AND (NOTE.scope = "entry-point")
', ' AND (NOTE.shift = "'+SC1,'"
', ' AND (PAGE.lhs = "'+A,'"
', ' AND (PAGE.rhs = "'+V,'")
']), SQL).

aphlbs_semantics(action('ANNOTATION_THEN_DO', annotation_update
(update_operation('DELETE'), annotation('ANNOTATION',
note(scope(page_part(relevant_chunk
(shift(signed_integer('+', I)), 'CHUNK'))),
relevance_multi_part(shift(signed_integer('+', SC1)), 'EXIT'))),
attribute_assignment(A, V))))), SQL) :-
concat_atom(['(DELETE * FROM NOTE
', ' WHERE NOTE.page IN
', ' (SELECT page FROM SELECTEDPAGES)
', ' WHERE SELECTEDPAGES.shift ="'+I,'"
', ' AND (NOTE.scope = "exit-point")
', ' AND (NOTE.shift = "'+SC1,'"
', ' AND (PAGE.lhs = "'+A,'"
', ' AND (PAGE.rhs = "'+V,'")
']), SQL).
aphlbs_semantics(action('ANNOTATION_THEN_DO', annotation_update
(update_operation('DELETE'), annotation('ANNOTATION',
note(scope(page_part(relevant_chunk
(shift(signed_integer('+', I)), 'CHUNK'),
relevant_single_part('CONTENT'))),
attribute_assignment(A, V))))), SQL) :-
concat_atom(['(DELETE * FROM NOTE 
WHERE NOTE.page IN 
(SELECT page FROM SELECTEDPAGES) 
WHERE SELECTEDPAGES.shift ="',I,") 
AND (NOTE.scope = "c-spec")
AND (PAGE.lhs = "',A,")
AND (PAGE.rhs = "',V,"))
], SQL).

aphlbs_semantics(action('ANNOTATION_THEN_DO', annotation_update
(update_operation('DELETE'), annotation('ANNOTATION',
note(scope(page_part(relevant_chunk
(shift(signed_integer('+', I)), 'CHUNK'),
relevant_single_part('RENDERING'))),
attribute_assignment(A, V))))), SQL) :-
concat_atom(['(DELETE * FROM NOTE 
WHERE NOTE.page IN 
(SELECT page FROM SELECTEDPAGES) 
WHERE SELECTEDPAGES.shift ="',I,") 
AND (NOTE.scope = "r-spec")
AND (PAGE.lhs = "',A,")
AND (PAGE.rhs = "',V,"))
], SQL).

aphlbs_semantics(action('ANNOTATION_THEN_DO', annotation_update
(note_projection(projection_operation('DROP_IF'),
condition_on_note(atomic_condition_on_note
(relevant_note(shift(signed_integer('+', I)), 'NOTE'),
284
aphlbs_semantics(action('ANNOTATION_THEN_DO', annotation_update
(note_projection(projection_operation('RETAIN_IF'),
condition_on_note(atomic_condition_on_note
(relevant_note(shift(signed_integer('+', I)), 'NOTE'),
containment_expression('CONTAINS', S))))), SQL) :-
concat_atom(['(DELETE * FROM NOTE 
', ' WHERE NOTE.page IN 
', ' (SELECT page FROM SELECTEDPAGES 
', ' AND SELECTEDPAGES.shift = ", I, ") 
', ' AND (NOTE.chunk IN 
', ' (SELECT CHUNK.chunk FROM CHUNK 
', ' WHERE CHUNK.chunk IN 
', ' (SELECT NOTE.chunk FROM NOTE 
', ' WHERE NOTE.n_string LIKE ", S, ")))) 
', ' WHERE NOTE.n_string LIKE ", S, ")))) 
', ' SQL). % concat all of above, put result in SQL

aphlbs_semantics(action('ANNOTATION_THEN_DO', annotation_update
(update_operation('INSERT'), annotation('ANNOTATION',
note(scope('PAGE'), attribute_assignment(A, V))))), SQL) :-
concat_atom(['<CF QUERY "Getselectedpages" >
', ' SELECT page, chunk, shift FROM SELECTEDPAGES
', ' '</CFQUERY>
', ' 
', ' <CF LOOP "Getselectedpages" >
', ' '</CFQUERY>
', ' 
', ' INSERT #NewID#, #Getselectedpages.page#, "null","attrib",
', ' INTO NOTE
', ' '</CFQUERY>
', ' '</CF LOOP>, SQL).
aphlbs_semantics(action('ANNOTATION_THEN_DO', annotation_update( update_operation('INSERT'), annotation('ANNOTATION', note(scope('PAGE'), attribute_assignment(A, V)))))), SQL) :-
  concat_atom(['<CF QUERY "Getselectedpages" >
', '  SELECT page, chunk, shift FROM SELECTEDPAGES
', '  <CF LOOP "Getselectedpages" >
', '  (INSERT #NewID#, #Getselectedpages.page#, "null",
   "attrib", "page","null", \n   ",",A,"", ",",V,"", "null"\n   
   INTO NOTE
   
   </CF QUERY>
', '  </CF LOOP>'], SQL).

aphlbs_semantics(action('HP_THEN_DO', hp_update (hp_terminal_rewrite(rewrite(scope('PAGE'), from(F), to(T)))))), SQL) :-
  concat_atom(['<CF QUERY "Getselectedpages" >
', '  SELECT page, chunk, shift FROM SELECTEDPAGES
', '  <CF LOOP "Getselectedpages" >
', '  (INSERT #NewID#, #Getselectedpages.page#, "null",
   "rewrite",
   "page","null,"F","",T,"", "null",
   
   INTO NOTE
   
   </CF QUERY>
', '  </CF LOOP>'], SQL).

aphlbs_semantics(action('ANNOTATION_THEN_DO', annotation_update( update_operation('INSERT'), annotation('ANNOTATION', note(scope('PAGE'), attribute_assignment(A, V)))))), SQL) :-
  concat_atom(['<CF QUERY "Getselectedpages" >
', '  SELECT page, chunk, shift FROM SELECTEDPAGES
', '  <CF LOOP "Getselectedpages" >
', '  (INSERT #NewID#, #Getselectedpages.page#, "null",
   "attrib", "page","null", \n   ",",A,"", ",",V,"", "null"\n   
   INTO NOTE
   
   </CF QUERY>
', '  </CF LOOP>'], SQL).

aphlbs_semantics(action('HP_THEN_DO', hp_update (hp_terminal_rewrite(rewrite(scope('PAGE'), from(F), to(T)))))), SQL) :-
  concat_atom(['<CF QUERY "Getselectedpages" >
', '  SELECT page, chunk, shift FROM SELECTEDPAGES
', '  <CF LOOP "Getselectedpages" >
', '  (INSERT #NewID#, #Getselectedpages.page#, "null",
   "rewrite",
   "page","null,"F","",T,"", "null",
   
   INTO NOTE
   
   </CF QUERY>
', '  </CF LOOP>'], SQL).
aphlbs_semantics(action('HP_THEN_DO', hp_update
(hp_terminal_rewrite(rewrite(scope(page_part
(relevant_chunk(SC1, 'CHUNK'), relevant_multi_part
(shift(signed_integer('+', I)), 'ENTRY')))),
from(F), to(T)), SQL) :-
concat_atom(["<CF QUERY "Getselectedpages" >\n',
  ' SELECT page, chunk, shift FROM SELECTEDPAGES \n',
  ' WHERE shift="", SC1, "\n',
  '</CFQUERY>\n',
  ' \n',
  ' <CF LOOP "Getselectedpages" >\n',
  '</CFQUERY>\n',
  ' (INSERT #NewID#, #Getselectedpages.page#, chunk,"rewrite", \n',
  ' "entry","",I,"","",F,"","",T,"" "null", \n',
  ' INTO NOTE
',
  '</CFQUERY>\n',
  '</CF LOOP>)\n'], SQL).

aphlbs_semantics(action('HP_THEN_DO', hp_update
(hp_terminal_rewrite(rewrite(scope(page_part
(relevant_chunk(SC1, 'CHUNK'), relevant_multi_part
(shift(signed_integer('+', I)), 'EXIT')))),
from(F), to(T)), SQL) :-
concat_atom(["<CF QUERY "Getselectedpages" >\n',
  ' SELECT page, chunk, shift FROM SELECTEDPAGES \n',
  ' WHERE shift="", SC1, "\n',
  '</CFQUERY>\n',
  ' \n',
  ' <CF LOOP "Getselectedpages" >\n',
  '</CFQUERY>\n',
  ' (INSERT #NewID#, #Getselectedpages.page#, chunk,"rewrite", \n',
  ' "exit","",I,"","",F,"","",T,"" "null", \n',
  ' INTO NOTE
',
  '</CFQUERY>\n',
  '</CF LOOP>)\n'], SQL).
aphlbs_feedback(Tree, batch, Reply) :-
    !,
    Reply = 'not_applicable',
    aphlbs_tree_print(Tree).
aphlbs_feedback(Tree, _Mode, Reply) :-
    aphlbs_ask(Tree, Reply).

aphlbs_ask(Tree, Reply) :-
    (yesno( 'do you want to see the tree')
        -> Reply = 'y', aphlbs_tree_print(Tree)
        ; Reply = 'n', write('print tree not requested.'), nl).

aphlbs_tree_print(Tree) :-
    write('this is the term:'),
    nl, write(Tree), nl,
    write('This is the tree:'),
    nl, tree_print(Tree).

/*

tree_print(Term) prints term in tree-fashion, indented by two.
*/
tree_print(Term) :-
    nl,
    tree_print_sub(Term,0,'-:'),!.

tree_print_sub(Term, Depth, Prefix) :-
    is_printable_list(Term)
    -> print_prefix(Depth, Prefix), name(Atom, Term), write(Atom), nl
    ; fail.

tree_print_sub(Term, Depth, Prefix) :-
    print_prefix(Depth, Prefix),
    (var(Term) -> (Name = Term, Arglist = []);
        Term =.. [Name | Arglist]),
    print(Name), nl,
    Depth_plus is Depth+1,
    print_args(Arglist, Depth_plus, Prefix).
is_printable_list([X]) :- !, is_printable(X).
is_printable_list([H|T]) :- is_printable(H), is_printable_list(T).

is_printable(X) :- % broadly speaking
    X >= 32,
    X =< 122.

print_prefix(0, _).
print_prefix(Depth, Prefix) :-
    Depth > 0,
    print(Prefix),
    Depth_minus is Depth-1,
    print_prefix(Depth_minus, Prefix).

print_args([], _Depth, _).
print_args([First | Rest], Depth, Prefix) :-
    tree_print_sub(First, Depth, Prefix),
    print_args(Rest, Depth, Prefix).
D.3  Sample Personalisation Request Text File

select_page_if
   page contains '1997'
hp_then_do{
   delete[2,chunk]
}
.
select_page_if
   true
hp_then_do{
   retain_if [2, chunk] contains '1998'
}
.
select_page_if
   true
hp_then_do{
   drop_if [1, chunk (1,entry)] contains 'Newlinks'
}
.
select_page_if
   true
an_then_do{
   insert annotation{
      page : level := 1;
   }
}
.
select_page_if
   true
an_then_do{
   delete annotation{
      [1, chunk] : level := 1;
   }
}
.

/*@ */
D.4 Personalisation Requests Program Run

Welcome to SWI-Prolog (Version 3.1.2)
Copyright (c) 1993-1998 University of Amsterdam. All rights reserved.

Adaptive, Personalizable Hyperlink-Based System
Version 0.0, Tue Nov 24 10:24:30 GMT 1998.

Beginning setup ...
Setup complete.

aphlbs? read(personalization_request, example_pr1).

personalization_request parsing succeeded
this is the term:
personalization_request(action_scope(SELECT_PAGE_IF, condition
(atomic_condition(atomic_condition_on_page
(PAGE, containment_expression(CONTAINS, 1997))))),
.(action(HP_THEN_DO, hp_update(hp_structure_update
(DELETE, page_part(relevant_chunk(shift(signed_integer(+, 2)), CHUNK))))))
This is the tree:

personalization_request
  -:action_scope
  --:-:SELECT_PAGE_IF'
  -=:condition
  --:-:atomic_condition
  -=:-:atomic_condition_on_page
  --:-:-:PAGE'
  -=:-:-:containment_expression
  --:-:-:-:CONTAINS'
  -=:-:-:-:1997
  :=:
  -=:action
  -=:-:HP_THEN_DO'
  -=:-:hp_update
  -=:-:hp_structure_update
  -=:-:-:DELETE'
  -=:-:-:page_part

291
INSERT PAGE.page, PAGE.chunk, PAGE.shift INTO SELECTEDPAGES
WHERE PAGE.page IN (SELECT PAGE.page FROM PAGE)
AND (PAGE.page IN
(SELECT PAGE.page FROM PAGE
AND PAGE.chunk IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.entrypointset IN
(SELECT ENTRYPINTSET.entrypointset FROM ENTRYPINTSET
WHERE ENTRYPINTSET.entrypoint IN
(SELECT ENTRYPINT.entrypoint FROM ENTRYPINT
WHERE ENTRYPINT.e_string LIKE "1997"))))

OR PAGE.page IN
(SELECT PAGE.page FROM PAGE
WHERE PAGE.chunk IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.cspec IN
(SELECT CSPEC.cspec FROM CSPEC
WHERE CSPEC.c-element IN
(SELECT C-ELEMENT.c-element FROM C-ELEMENT
WHERE C-ELEMENT.c_string LIKE "1997")))

OR PAGE.page IN
(SELECT PAGE.page FROM PAGE
WHERE PAGE.chunk IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.rspec IN
(SELECT RSPEC.rspec FROM RSPEC
WHERE RSPEC.r-element IN
(SELECT R-ELEMENT.r-element FROM R-ELEMENT
WHERE R-ELEMENT.r_string LIKE "1997")))

OR PAGE.page IN
(SELECT PAGE.page FROM PAGE
WHERE PAGE.chunk IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.rspec IN
(SELECT RSPEC.rspec FROM RSPEC
WHERE RSPEC.r-element IN
(SELECT R-ELEMENT.r-element FROM R-ELEMENT
WHERE R-ELEMENT.r_string LIKE "1997")))
WHERE CHUNK.exitpointset IN
(SELECT EXITPOINTSET.entrypointset FROM EXITPOINTSET
WHERE EXITPOINTSET.exitpoint IN
(SELECT EXITPOINT.exitpoint FROM EXITPOINT
WHERE EXITPOINT.x_string LIKE "1997")))
(DEDELETE * FROM PAGE
WHERE PAGE.page IN
(SELECT page FROM SELECTEDPAGES)
AND (PAGE.shift = "2")

personalization_request parsing succeeded
this is the term:
personalization_request(action_scope(SELECT_PAGE_IF, condition
(atomic_condition(TRUE))), ..(action(HP_THEN_DO, hp_update
(hp_structure_update(projection_operation(RETAIN_IF),
condition_on_hp_construct(atomic_condition_on_hp_construct
(page_part(relevant_chunk(shift(signed_integer(+, 2)), CHUNK)),
containment_expression(CONTAINS, 1998)))))))
This is the tree:

personalization_request
  |--action_scope
  |  |--:SELECT_PAGE_IF'
  |  |--condition
  |     |--:atomic_condition
  |     |     |--:TRUE'
  |     |     |--..
  |  |--action
  |     |--:HP_THEN_DO'
  |     |--hp_update
  |     |     |--:hp_structure_update
  |     |     |     |--:projection_operation
  |     |     |     |     |--:RETAI'If'
  |     |     |     |     |--:condition_on_hp_construct
  |     |     |     |     |     |--:attribute_on_hp_construct
  |     |     |     |     |     |     |--:page_part
  |     |     |     |     |     |     |     |--:relevant_chunk
  |     |     |     |     |     |     |     |--:shift
  |     |     |     |     |     |     |     |     |--:signed_integer
  |     |     |     |     |     |     |     |     |     |--:+
  |     |     |     |     |     |     |     |     |     |     |--:2
  |     |     |     |     |     |     |     |     |     |     |     |--:'CHUNK'
-:--:--:--:--:--:containment_expression
-:--:--:--:--:--:'CONTAINS'
-:--:--:--:--:--:1998

INSERT PAGE.page, PAGE.chunk, PAGE.shift INTO SELECTEDPAGES
WHERE PAGE.page IN (SELECT PAGE.page FROM PAGE)
AND (PAGE.page IN
(select page from PAGE))

DELETE * FROM PAGE
WHERE PAGE.page IN
(SELECT page FROM SELECTEDPAGES)
AND PAGE.chunk NOT IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.entrypointset IN
(SELECT ENTRYPONTSET.entrypointset FROM ENTRYPONTSET
WHERE ENTRYPONTSET.entrypoint IN
(SELECT ENTRYPONT ENTRYPOINT FROM ENTRYPONT
WHERE ENTRYPONT.e_string LIKE "1998"))
AND (PAGE.shift = "2")

OR PAGE.page IN
(SELECT page FROM SELECTEDPAGES)
AND PAGE.chunk NOT IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.cspec IN
(SELECT CSPEC.spec FROM CSPEC
WHERE CSPEC.c-element IN
(SELECT C-ELEMENT.c-element FROM C-ELEMENT
WHERE C-ELEMENT.c_string LIKE "1998")
AND (PAGE.shift = "2")

OR PAGE.page IN
(SELECT page FROM SELECTEDPAGES)
AND PAGE.chunk NOT IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.rspec IN
(SELECT RSPEC.spec FROM RSPEC
WHERE RSPEC.r-element IN
(SELECT R-ELEMENT.r-element FROM R-ELEMENT
WHERE R-ELEMENT.r_string LIKE "1998")
AND (PAGE.shift = "2")
OR  PAGE.page IN
(SELECT page FROM SELECTEDPAGES)
AND  PAGE.chunk NOT IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.exitpointset IN
(SELECT EXITPOINTSET.entrypointset FROM EXITPOINTSET
WHERE EXITPOINTSET.exitpoint IN
(SELECT EXITPOINT.exitpoint FROM EXITPOINT
WHERE EXITPOINT.x_string LIKE "1998")))
AND (PAGE.shift = "2")

personalization_request parsing succeeded
this is the term:
personalization_request(action_scope(SELECT_PAGE_IF, condition
(atomic_condition(TRUE))), ..(action(HP_THEN_DO, hp_update
(hp_structure_update(projection_operation(DROP_IF),
condition_on_hp_construct(atomic_condition_on_hp_construct
(page_part(relevant_chunk(shift(signed_integer(+, 1)), CHUNK),
relevant_multi_part(shift(signed_integer(+, 1)), ENTRY)),
containment_expression(CONTAINS, Newlinks))))))))
This is the tree:

personalization_request
  :action_scope
    :SELECT_PAGE_IF
      :condition
        :atomic_condition
          :TRUE
            ...
              :action
                :HP_THEN_DO
                  :hp_update
                    :hp_structure_update
                      :projection_operation
                        :DROP_IF
                          :condition_on_hp_construct
                            :atomic_condition_on_hp_construct
                              :page_part
                                :relevant_chunk
                                  :shift
                                    :signed_integer
                                      +
INSERT PAGE.page, PAGE.chunk, PAGE.shift INTO SELECTEDPAGES
WHERE PAGE.page IN (SELECT PAGE.page FROM PAGE)
AND (PAGE.page IN
(select page from PAGE))

(DELTER * FROM PAGE
WHERE PAGE.page IN
(SELECT page FROM SELECTEDPAGES)
AND PAGE.chunk IN
(SELECT CHUNK.chunk FROM CHUNK
WHERE CHUNK.entrypointset IN
(SELECT ENTRYPONTSET.entrypointset FROM ENTRYPONTSET
WHERE ENTRYPONTSET.entrypoint IN
(SELECT ENTRYPONT.entrypoint FROM ENTRYPONT
WHERE ENTRYPONT.e_string LIKE "Newlinks")))
AND (ENTRYPOINT.shift ="1")
AND (PAGE.shift ="1")

personalization_request parsing succeeded
this is the term:
personalization_request(action_scope(SELECT_PAGE_IF,
condition(atomic_condition(TRUE))), ..(action(ANNOTATION_THEN_DO,
annotation_update(update_operation(INSERT), annotation(ANNOTATION,
note(scope(PAGE), attribute_assignment(LEVEL, 1)))))))
This is the tree:

personalization_request
  :action_scope
  :SELECT_PAGE_IF
  :condition
    :atomic_condition
personalization_request parsing succeeded

this is the term:

```
personalization_request(action_scope(SELECT_PAGE_IF,
    condition(atomic_condition(TRUE))), ..(action(ANNOTATION_THEN_DO,
    annotation_update(update_operation(DELETE), annotation(ANNOTATION,
    note(scope(page_part(relevant_chunk(shift(signed_integer(+, 1)), CHUNK))),
    attribute_assignment(LEVEL, 1))))))
```

This is the tree:
```
personalization_request
  -:action_scope
```

```
INSERT PAGE.page, PAGE.chunk, PAGE.shift INTO SELECTEDPAGES
WHERE PAGE.page IN (SELECT PAGE.page FROM PAGE)
AND (PAGE.page IN
(select page from PAGE))

<CF QUERY "Getselectedpages" >
      SELECT page, chunk, shift FROM SELECTEDPAGES
<CFQUERY>

<CF LOOP "Getselectedpages" >
</CFQUERY>

(INSERT #NewID#, #Getselectedpages.page#, "null","attrib",
"page","null", "LEVEL", "1" INTO NOTE
</CFQUERY>
</CF LOOP>)

personalization_request parsing succeeded

this is the term:

```
personalization_request(action_scope(SELECT_PAGE_IF,
    condition(atomic_condition(TRUE))), ..(action(ANNOTATION_THEN_DO,
    annotation_update(update_operation(DELETE), annotation(ANNOTATION,
    note(scope(page_part(relevant_chunk(shift(signed_integer(+, 1)), CHUNK))),
    attribute_assignment(LEVEL, 1))))))
```

This is the tree:
```
personalization_request
  -:action_scope
-:-:'SELECT_PAGE_IF'
-:-:condition
-:-:-:atomic_condition
-:-:-:-:'TRUE'
-:-
-:-:action
-:-:-:'ANNOTATION_THEN_DO'
-:-:-:annotation_update
-:-:-:-:update_operation
-:-:-:-:'DELETE'
-:-:-:-:annotation
-:-:-:-:-:'ANNOTATION'
-:-:-:-:-:note
-:-:-:-:-:scope
-:-:-:-:-:-:page_part
-:-:-:-:-:-:relevant_chunk
-:-:-:-:-:-:shift
-:-:-:-:-:-:signed_integer
-:-:-:-:-:-:-:+
-:-:-:-:-:-:1
-:-:-:-:-:-:-:'CHUNK'
-:-:-:-:-:-:attribute_assignment
-:-:-:-:-:-:'LEVEL'
-:-:-:-:-:-:-:1
    INSERT PAGE.page, PAGE.chunk, PAGE.shift INTO SELECTEDPAGES
    WHERE PAGE.page IN (SELECT PAGE.page FROM PAGE)
    AND (PAGE.page IN
    (select page from PAGE))

    (DELETE * FROM NOTE
    WHERE NOTE.page IN
    (SELECT page FROM SELECTEDPAGES
    WHERE SELECTEDPAGES.shift ="1")
    AND (NOTE.scope = "chunk")
    AND (PAGE.lhs = "LEVEL")
    AND (PAGE.rhs = "1"))
done

aphlbs? Quit requested.
Yes
Appendix E
PAS Front-End Cold Fusion
Application Programs

E.1 A WWW-based Application for the Dynamic Generation of Hyperpages

<!--- Definition -->
<!--- This script generates a hyperpage from a sequence of SQL statements-->
<!--- which are used to retrieve data and present it as a sequence of -->
<!--- hyperpage component parts (entry points, chunks, exit points). -->
<!--- Each of which may or may not have an associated annotation which -->
<!--- may be accessed from this page. -->
<!--- Inputs : Hyperpage_Reference --->
<!--- Output : Annotated Hyperpage -->
<!--- This script written in JavaScript is used to display a popup window -->
<!--- containing annotations for any component of the hyperpage. -->

<SCRIPT LANGUAGE="Javascript">
var popup_window = null;
function popup(status,url) {
   if(status != 0) {
      if(popup != null) popup.focus();
      else {
         var popup = open(url, "Popup","width=300,height=150,scrollbars=yes");
         popup_window = popup;
      }
   } else {
      if(popup_window != null) popup_window.close();
   }
}
</SCRIPT>
<CFSET AnnotationType = ""> <!--- Variable to hold Annotation Type --->
<CFSET AnnotationPage = 0> <!--- Page Variable for Current Annotation --->
<CFSET Annotationchunk = 0> <!--- Chunk Variable for Current Annotation --->

<CFQUERY NAME="GetChunksForPage" DATASOURCE="model1">
SELECT P.page, C.chunk, P.shift
FROM Pages P, Chunks C
WHERE P.chunk = C.chunk <!---Join Pages to Chunks--->
AND P.page = 10 <!---Current hyperpage reference--->
ORDER BY P.shift
</CFQUERY>

<CFSET AnnotationType = "Page">
<CFSET AnnotationPage = #GetChunksForPage.Page#>
The following SQL statement is used to test for the presence of an annotation note for the given page.

**Inputs:** page, Annotation Type

**Output:** page, Annotation Type chunk

```sql
<CFQUERY NAME="CheckForAnnotation" DATASOURCE="model1">
SELECT A.page, S.type, S.chunk
FROM SinglePartAnnotationNotes S, Annotations A
WHERE A.note = S.note
AND (A.page = #AnnotationPage#)
AND (S.type = '#AnnotationType#')
</CFQUERY>
```

If the above SQL statement returns a record place a pin image at the end of the current page part.

```cfif #CheckForAnnotation.RecordCount# IS NOT 0>
<CFOUTPUT>
<A HREF="" onMouseover="popup(1, 'frontendfrontendpopup.cfm?ICurrentPage=#AnnotationPage#&ICurrentChunk=#AnnotationPage#&SCurrentType=#AnnotationType#')">
<img SRC="pin1.gif" BORDER=0></A>
</CFOUTPUT>
</cfif>

Loop previously prepared SQL statement to get the all chunks for a given page until there are no more chunks.

```cfloop query="GetChunksForPage"```
<CFQUERY NAME="GetChunk" DATASOURCE="model1">
SELECT    chunk, entrypointset, cspec, rspec, exitpointset
FROM       Chunks
WHERE       chunk = #GetChunksForPage.chunk#
</CFQUERY>

<!---  ***************************************************************** --->
<!--- The following SQL statement is used to get all entry points for the --->
<!--- current Chunks entrypoint shift --->
<!--- Inputs : entrypointset --->
<!--- Output : entrypoint --->

<CFQUERY NAME="GetEntryPointsForChunk" DATASOURCE="model1">
SELECT    entrypointset, Entrypoint, shift
FROM       Entrypoints
WHERE       entrypointset = #GetChunk.entrypointset#
ORDER BY    shift
</CFQUERY>

<!---  ***************************************************************** --->
<!--- The following SQL statement is used to get all assignment ID's for the --->
<!--- current chunks cspec. --->
<!--- Inputs : cspec --->
<!--- Output : cspec, assignment --->

<CFQUERY NAME="GetAssignments" DATASOURCE="model1">
SELECT    cspec, assignment
FROM       Cspecs
WHERE       cspec = #GetChunk.cspec#
</CFQUERY>

<!---  ***************************************************************** --->
<!---  ***************************************************************** --->

<!--- The following SQL statement is used to get all template ID's for the  --->
<!--- current Chunks Rspec.  --->
<!--- Inputs : Rspec  --->
<!--- Output : Rspec , template  --->

<CFQUERY NAME="GetTemplates" DATASOURCE="model1">
SELECT rspec, template
FROM Rspecs
WHERE rspec = #GetChunk.rspec#
</CFQUERY>

<!---  ***************************************************************** --->

<!--- The following SQL statement is used to get all Exit points for the --->
<!--- current Chunks Exitpointset set.  --->
<!--- Inputs : Exitpointset  --->
<!--- Output : Exitpointset , Exitpoint, shift  --->

<CFQUERY NAME="GetExitPointsForChunk" DATASOURCE="model1">
SELECT exitpointset, Exitpoint, shift
FROM Exitpoints
WHERE exitpointset = #GetChunk.exitpointset#
ORDER BY shift
</CFQUERY>

<!---  ***************************************************************** --->

<!--- Loop Previously prepared query to output all entry points for the --->
<!--- current chunks entry point set until there are no more entry points.  --->

<cfloop query="GetEntryPointsForChunk">
<!--- Output all entry points for a given chunk --->
<cfoutput>
<P><A Name="#Entrypoint#">#Entrypoint#</A></cfoutput>
</cfoutput>
</cfloop>

<!---  ***************************************************************** --->

303
<!--- Assign variables values for the current Entrypoint sets annotation  --->

<CFSET AnnotationType = "Entry">
<CFSET AnnotationPage = #GetChunksForPage.page#>
<CFSET Annotationchunk = #GetChunk.chunk#>

<!---  ***************************************************************** --->
<!--- The following SQL statement is used to test for the presence of an --->
<!--- annotation note for the given Entrypoint sets. --->
<!--- Inputs : Page, Annotation Type, Chunk --->
<!--- Output : Page, Annotation Type Chunk --->

<CFQUERY NAME="CheckForAnnotation" DATASOURCE="model1">
SELECT A.page, S.type, S.chunk
FROM SinglePartAnnotationNotes S, Annotations A
WHERE A.note = S.note
AND (A.page = #AnnotationPage#)
AND (S.type = '#AnnotationType#')
AND (S.Chunk = #Annotationchunk#)
</CFQUERY>

<!---  ***************************************************************** --->
<!--- If the above SQL statement returns a record place a pin image at --->
<!--- the end of the current entry point set. --->

<CFIF #CheckForAnnotation.RecordCount# IS NOT 0>
<CFOUTPUT>
<A HREF="" onMouseover="popup(1,'frontendpopup.cfm?
  ICurrentPage=#AnnotationPage#&ICurrentChunk=#Annotationchunk#
  &SCurrentType=#AnnotationType#')">
  <IMG SRC="pin1.gif" BORDER=0></A>
</CFOUTPUT>
</cfif>
</p>

<!---  ***************************************************************** --->
<!--- Initialise Variables --->

<CFSET CurrentContentIdentifier =""
<!--- Variable used to assign content expression to assignment identifier --->
<CFSET SVarName =""
<cfset IStart =0

<!--- Loop Previously prepared query to get all assignments IDs for the current chunks Cspec until there are no more assignments

<cfloop query="GetAssignments">
<!--- SQL statement is used to get all assignment data for the current assignment. 
<!--- Inputs : assignment 
<!--- Output : Assignment , Identifier, Content Exp, Contenttype

<CFQUERY NAME="GetAssignment" DATASOURCE="model1">
SELECT assignment, identifier, content_Exp, Contenttype
FROM Assignments
WHERE (assignment = #GetAssignments.assignment#)
</CFQUERY>

<!--- SQL statement is used to get all template data for the current Chunks Rspec. 
<!--- Inputs : Template 
<!--- Output : Template , Style , TemplateElement, TemplateType

<CFQUERY NAME="GetTemplate" DATASOURCE="model1">
SELECT template, style, templateelement, templatetype
FROM template
WHERE (template = #GetTemplates.template#)
</CFQUERY>

<!--- --->

305
<cfloop query="GetTemplate">

<!--- If the Template is an assignment then --->

<CFIF #GetTemplate.TemplateType# EQ "Assignment">

<!--- If the identifier in the assignment is the same as the template element then--->

<CFIF #GetTemplate.templateelement# EQ #GetAssignment.identifier#>

<!---Create a variable to hold Assignment Identifier --->

<!---Assign the assignments content expression to it --->

<CFSET CurrentContentIdentifier=SetVariable("#GetTemplate.TemplateElement#", "#GetAssignment.Content_Exp#")>

<!---Get the template style for the current template --->

<CFQUERY NAME="GetStyle" DATASOURCE="model1">
SELECT style, LHS, RHS
FROM Styles
WHERE (style = #GetTemplate.style#)
</CFQUERY>
</CFIF>
</CFIF>
</cfloop>
<CFIF #GetAssignment.contenttype# IS "Value">
<!--- Write the current template with its formatting --->
<cfoutput query = "GetAssignment" group="Assignment">
#GetStyle.LHS##CurrentContentIdentifier##GetStyle.RHS#
</cfoutput>
</CFIF>

<CFELSEIF #GetAssignment.contenttype# IS "DBS-String">
<!--- Prepare the assignments DBS-String --->
<CFQUERY NAME="DynamicQuery" DATASOURCE="model1">
#CurrentContentIdentifier#
</CFQUERY>

<!--- Run the query until it has fetched all the data --->
<cfloop query="DynamicQuery">
<!--- As the fields to output are unknown find out what they are and get index for it --->
<cfloop list="#DynamicQuery.ColumnList#" index="CurrField">
<!--- Output the current result sets field along with its formating --->
<cfoutput>
#GetStyle.LHS##Evaluate("DynamicQuery.#CurrField#")##GetStyle.RHS#
</cfoutput>
</cfloop>
</cfloop>
</cfif>
</cfloop>
</cfif>

<!---  ********************************************************************************  --->
<!--- Loop Previously prepared query to get all template data --->
<!--- for a given chunks Rspec until there are no more templates --->
<cfloop query="GetTemplate">
<!--- If the given Cspecs current assignemt is a UIS-String run this --->
<CFIF #GetTemplate.TemplateType# IS "UIS-String">
<!---Get the template style for the current template --->
<CFQUERY NAME="GetStyle" DATASOURCE="model1">
SELECT style, LHS, RHS
FROM Styles
WHERE (style = #GetTemplate.style#)
</CFQUERY>
<!--- Output the current result with its formating --->
<CFOUTPUT>
#GetStyle.LHS##GetTemplate.TemplateElement##GetStyle.RHS#
</CFOUTPUT>
</CFIF>
</cfloop>
<!--- Assign variables values for the current Chunks annotation --->
<CFSET AnnotationType = "Chunk">
<CFSET AnnotationPage = #GetChunksForPage.page#>
<CFSET Annotationchunk = #GetChunk.chunk#>

<!--- The following SQL statement is used to test for the precense of an --->
<!--- annotation note for the given Entrypoint sets. --->
<!--- Inputs : Page , Annotation Type, Chunk --->
<!--- Output : Page , Annotation Type Chunk --->

<CFQUERY NAME="CheckForAnnotation" DATASOURCE="model1">
SELECT annotations.page, SinglePartAnnotationNoteS.type, SinglePartAnnotationNoteS.chunk
FROM SinglePartAnnotationNotes, annotations
WHERE annotationS.note = SinglePartAnnotationNoteS.note
AND (annotations.page = #AnnotationPage#)
AND (SinglePartAnnotationNoteS.Chunk = #Annotationchunk#)
AND (SinglePartAnnotationNoteS.type = '#AnnotationType#')
</CFQUERY>
<!--- If the above SQL statement returns a record place a pin image at the--->
<!--- end of the current entry point set. --->

<CFIF #CheckForAnnotation.RecordCount# IS NOT 0>
<!--- If there is place a pin image at the current point --->
<CFOUTPUT>
<A HREF="onMouseover="popup(1,'frontendpopup.cfm?
  ICurrentPage=#AnnotationPage#&ICurrentChunk=#Annotationchunk#
  &SCurrentType=#AnnotationType#')">
  <IMG SRC="pin1.gif" BORDER=0></A>
</CFOUTPUT>
</CFIF>

<!--- Loop Previously prepared query to output all exit points for --->
<!--- the current chunks exit point set. --->
<cfloop query="GetExitPointsForChunk">

<!--- Output all the exit points for a given chunk --->
<cfoutput>
<P>#Exitpoint#</cfoutput>

<!--- Assign variables values for the current exit point sets annotation --->

<CFSET AnnotationType = "Exit">
<CFSET AnnotationPage = #GetChunksForPage.page#>
<CFSET Annotationchunk = #GetChunk.chunk#>
<CFSET IExit = #GetExitPointsForChunk.shift#>

<!--- The following SQL statement is used to test for the precense of an --->
<!--- annotation note for the given exitpoint sets. --->
<!--- Inputs : Page, Annotation Type, Chunk --->
<!--- Output : Page, Annotation Type Chunk --->

<CFQUERY NAME="CheckForAnnotation" DATASOURCE="model1">
SELECT annotations.page, MultiPartAnnotationNoteS.type, MultiPartAnnotationNoteS.chunk
FROM MultiPartAnnotationNotes, annotations
WHERE annotations.note = MultiPartAnnotationNoteS.note
AND (annotations.page = #AnnotationPage#)
AND (MultiPartAnnotationNoteS.type = '#AnnotationType#')
AND (MultiPartAnnotationNoteS.chunk = #Annotationchunk#)
</CFQUERY>
<!--- *********************** --------------------------------------------------->

<!--- If the above SQL statement returns a record place a pin image at the--->
<!--- end of the current entry point set. --->
<CFIF #CheckForAnnotation.RecordCount# IS NOT 0>
<CFOUTPUT>
<A HREF="onMouseover="popup(1,'frontendpopup.cfm?
ICurrentPage=#AnnotationPage#&ICurrentChunk=#Annotationchunk#
&SCurrentType=#AnnotationType#&ICurrentExitPoint=#IExit#')">
<IMG SRC="pin1.gif" BORDER=0></A>
</CFOUTPUT>
</CFIF>
</cfloop>
</cfloop>
E.2 A WWW-based Application for the Display of Hyperpage Annotations

<cfparam name="ICurrentPage" default=0>
<cfparam name="ICurrentchunk" default=0>
<cfparam name="SCurrenttype" default="">
<cfparam name="ICurrentExitPoint" default=0>

<CFIF (SCurrenttype IS "Page") OR (SCurrenttype IS "chunk")
OR (SCurrenttype IS "Entry")>
<CFQUERY NAME="modelgetsinglepartnoteid" DATASOURCE="model1">
SELECT A.note, AN.type
FROM Annotations A, SinglePartAnnotationNotes AN
WHERE A.page = #ICurrentPage#
AND A.note = AN.note
ORDER BY AN.type
</CFQUERY>
</CFIF>

<cfloop query="modelgetsinglepartnoteid">

<CFIF SCurrenttype IS "chunk" >
<CFQUERY NAME="getnote" DATASOURCE="model1">
SELECT note, type, chunk, LHS, RHS
FROM SinglePartAnnotationNotes
WHERE note = #modelgetsinglepartnoteid.note#
AND chunk = #ICurrentchunk#
AND (type = '#SCurrenttype#')
</CFQUERY>
</CFIF>

<CFELSEIF SCurrenttype IS "Entry" >
<CFQUERY NAME="getnote" DATASOURCE="model1">
SELECT note, type, chunk, LHS, RHS
FROM SinglePartAnnotationNotes
WHERE note = #modelgetsinglepartnoteid.note#
AND chunk = #ICurrentchunk#
AND (type = '#SCurrenttype#')
</CFQUERY>
</CFELSEIF>

<CFELSEIF SCurrenttype IS "Page" >
<CFQUERY NAME="getnote" DATASOURCE="model1"/>
</CFELSEIF>
SELECT note, type, chunk, LHS, RHS
FROM SinglePartAnnotationNotes
WHERE note = #modelgetsinglepartnoteid.note#
AND chunk = #ICurrentchunk#
AND (type = '#SCurrenttype#')
</CFQUERY>
</CFIF>

<cfoutput query="getnote">
<P>#type#: <P>#LHS# := '#RHS#'
</cfoutput>
</cfloop>

<CFELSEIF SCurrenttype IS "Exit">
<CFQUERY NAME="modelgetmultipartnoteid" DATASOURCE="model1">
SELECT A.note, AN.type
FROM Annotations A, MultiPartAnnotationNotes AN
WHERE A.page = #ICurrentPage#
AND A.note = AN.note
ORDER BY AN.shift, AN.LHS
</CFQUERY>

<cfloop query="modelgetmultipartnoteid">
<CFIF SCurrenttype IS "Exit">
<CFQUERY NAME="getnote" DATASOURCE="model1">
SELECT MPAN.note, MPAN.type, MPAN.chunk, MPAN.LHS, MPAN.RHS
FROM MultiPartAnnotationNotes MPAN, ExitPoints EP
WHERE MPAN.note = #modelgetmultipartnoteid.note#
AND MPAN.chunk = #ICurrentchunk#
AND (MPAN.type = '#SCurrenttype#')
AND MPAN.shift = EP.shift
AND EP.shift = #ICurrentExitPoint#
</CFQUERY>
</CFIF>

<cfoutput query="getnote" group="RHS">
<P>#type#: <P>#LHS# := '#RHS#'
</cfoutput>
</cfloop>
</CFELSEIF>
Appendix F
The Relational Data Model and Relational Algebra

This appendix defines the basic concepts which underpin the relational data model that are used to represent hyperpages and hyperpage annotations. This appendix also briefly describes the relational algebra used to operate on relations.

F.1 Basic Concepts of Relational Data Model

This section defines the basic concepts which underpin the relational data model. These are: domain, relation, attribute, tuple and keys. The definitions provided are adapted from [Mannila and Räihä, 1994, Abiteboul et al., 1995].

In the relational model, data is conceptually stored in tables. The columns in the tables are named, and are called attributes. Upper-case letters from the beginning of the alphabet (like A, B and C) will denote attributes when they occur in the definitions that follow. Attribute sets are denoted by upper-case letters from the end of the alphabet, such as X, Y and Z. A set consisting of a single attribute is denoted by the attribute itself.

F.1.1 Relational Data Model Definitions

Definition (Attribute Sequence). An attribute sequence is an ordered set of attributes.

Attribute sequences are written by enclosing the attributes in angle brackets, as in ⟨A₁, ..., Aₙ⟩. The same symbols are used to denote both attribute sets and attribute sequences. Thus X, Y, and Z can denote sequences, and attribute A can denote {A}. Attribute sequences can be used in set operations; then the ordering of the elements is ignored. Similarly, sets can be used as sequences by ordering the elements in some way, e.g., lexicographically.
Each attribute $A$ has an associated domain, denoted $\text{Dom}(A)$. If a table has an $A$-column, all values appearing in the column must belong to $\text{Dom}(A)$. For a set of attributes $X$, the domain of $X$ is defined as follows

**Definition (Domain).**

$$\text{Dom}(X) = \bigcup_{A \in X} \text{Dom}(A).$$

The rows in a table consist of values for all attributes labelling the columns of the table. Each row is called a tuple and the set of all tuples in a table is called a relation. Therefore if the attributes of a table are $A_1, \ldots, A_n$, the table is a subset of $\text{Dom}(A_1) \times \ldots \times \text{Dom}(A_n)$, that is a relation in the set-theoretical sense. Lower-case letters are used to denote tuples (e.g., $t$, $u$, $v$ and $w$) and relations (e.g., $r$ and $s$). Both domains and relations may be infinite sets.

**Definition (Relation Schema).** Let $r$ be a relation whose attributes are $A_1, \ldots, A_n$. Then $\{A_1, \ldots, A_n\}$ is the relation schema of $r$. In relational terms, $r$ is a relation over $\{A_1, \ldots, A_n\}$.

A relational schema is a term used to describe schemas expressed within the relational model.

It is useful to give names to relation schemas; upper-case SANS-SERIF letters are used for this purpose. Thus, if $R = \{A_1, \ldots, A_n\}$, $r$ is a relation over $R$. Furthermore $r$ is said to be an instance of $R$. It is also useful to indicate which attributes belong to a relation or relation schema. Thus $r(A_1, \ldots, A_n)$ and $R(A_1, \ldots, A_n)$ may be used instead of $r$ and $R$, respectively. Finally, it is useful to give names to attributes; lower-case italic sans-serif letters are used for this purpose.

**Definition (Tuple).** Let $r$ be a relation over the relation schema $R(A_1, A_2, \ldots, A_n)$, then $r$ is a set of n-tuples $r = \{t_1, t_2, \ldots, t_m\}$ where each n-tuple is an ordered list of values $t = <v_1, v_2, \ldots, v_n>$ and each value $v_i$, $1 \leq i \leq n$, is an element of $\text{Dom}(A_i)$ or is a null value. Therefore a tuple $t \in r$ is a mapping $t: R \mapsto \text{Dom}(R)$ such that $t(A) \in \text{Dom}(A)$ for each $A \in R$.

It is useful to give names to a value $v_i$ in a tuple $t$ for an attribute $A_i$; lower-case sans-serif letters are used for this purpose.

Since a relation is a set, all tuples in a relation are distinct, i.e., no two tuples in a relation have the same combination of values for all their attributes.

**Definition (Key).** A key of a relation schema is a set of attributes whose values are sufficient for identifying the tuples in the relation instance.

Note that the combination of all the attributes of a tuple is by definition always a key.
**Definition (Candidate Key).** A relation may have a subset of attributes which are unique for each tuple. Such a subset is called a candidate key.

It is common to designate one of the candidate keys of a relation to be the primary key of that relation.

**Definition (Primary Key).** A primary key is a candidate key used to identify tuples in the relation.

For convenience a further attribute may be included in a relation, whose role is to uniquely identify a tuple in a relation.

**Definition (Foreign Key).** A foreign key is a key which is included in a relation as a reference to another relation. A set of attributes $FK$ in a relation $R_1$ is a foreign key of $R_1$ if the attributes in $FK$ have the same domain as the primary key attributes $PK$ of another relation $R_2$.

The attributes $FK$ can be understood as a reference to the relation $R_2$. Furthermore a value of $FK$ in a tuple $t$ of $R_1$ must either occur as a value of $PK$ for some tuple $t$ in $R_2$ or be a null value.

### F.2 Relational Algebra

Relational algebra is a formalism for writing queries to databases that conform to the relational model. Relational algebra contains a set of basic operations on relations. In addition, several derived operations (operations that can be expressed as a sequence of basic operations) are also used. Relational algebraic operations can be categorised into retrievals which do not change the database state and updates which do. The core of relational algebra can be chosen in different ways. In this section one possibility is presented.

#### F.2.1 Retrieval Operations

Retrieval operations can be used, for example, to select tuples from relations or to combine related tuples from several relations. The result of each operation is a new relation which can be used in subsequent operations.

Retrieval operations may be divided into two groups: traditional set-theoretic operations (union, intersection and difference) and operations first proposed for the relational data model (SELECTION, PROJECTION, RENAMING and JOIN). Set-theoretic operations apply to the relational model because a relation is a set of tuples.
Set-Theoretic Operations

Conventional set operations union, intersection and difference (denoted by \( r \cup s \), \( r \cap s \) and \( r \setminus s \), respectively) are now defined.

**Definition (Union).**
\[
(r \cup s) = \{ t | t \in r \text{ or } t \in s \}
\]
The result of this operation is a relation that includes all tuples that are either in \( r \) or in \( s \) or in both.

**Definition (Intersection).**
\[
(r \cap s) = \{ t | t \in r \text{ and } t \in s \}
\]
The result of this operation is a relation that includes all tuples that are in both \( r \) and \( s \).

**Definition (Difference).**
\[
(r \setminus s) = \{ t | t \in r \text{ and } t \notin s \}
\]
The result of this operation is a relation that includes all tuples that are in \( r \) but not in \( s \).

*Remark.* Each of the above operations is binary. These operations assume that the two arguments have the same relation schema.

Relational Data Model Operations

*Projection* can be seen as the deletion and/or rearrangement of one or more “columns” of a relation. Let \( r \) be a relation over \( R \), and \( X \subseteq R \). For a tuple \( t \in r \) the projection of \( t \) on \( X \), denoted \( \pi_X(t) \), is defined as the mapping \( t' : X \rightarrow \text{Dom}(X) \) such that \( t'(A) = t(A) \) for each \( A \in X \) and \( t'(A) \) is null for each \( A \in R \setminus X \).

**Definition (PROJECTION).** The projection of relation \( r \) on \( X \) is defined as
\[
\pi_x(r) = \{ \pi_x(t) | t \in r \}
\]

*Selection* can be seen as selecting a subset of tuples in a relation. Selection is based on a condition which is a formula composed of constraints, attribute names, comparison operators, and the logical operators \( \lor \) (or), \( \land \) (and), and \( \neg \) (not). Satisfaction is defined in the standard way; for instance if \( \psi \) is “\( A = 5 \land B \leq 3 \)”, tuple \( t \) satisfies \( \psi \) if \( \pi_A(t) = 5 \) and \( \pi_B(t) \leq 3 \).

**Definition (SELECTION).** The selection operation based on a condition \( \psi \) is defined as
\[
\sigma_\psi(r) = \{ t \in r | \psi(t) \}.
\]
Renaming modifies the schema of the argument relation. The renaming operation is typically used to allow a relation to be used as an argument of one of the other operations, such as union or intersection. Let \( r \) be a relation over \( R \), and let \( A \in R \) and \( B \notin R \). Then \( r \) with \( A \) renamed to \( B \), denoted \( \rho_{A \rightarrow B}(r) \), is a relation over the attribute set \( (R \setminus \{A\}) \cup \{B\} \) and is defined as

**Definition (RENAMEING).**

\[ \rho_{A \rightarrow B}(r) = \{ t \mid \exists u \in r \text{ such that } t[B] = u[A] \text{ and } t[C] = u[C] \text{ if } C \neq B \} \]

The Cartesian product of two relations \( r \) over \( R \) and \( s \) over \( S \), is a relation with \( |R| + |S| \) columns and \( |r| \cdot |s| \) rows containing all the rows obtainable by placing a row of \( s \) at the end of a row of \( r \). Assume that \( R \) and \( S \) have no common attributes. For tuples \( t \in r \) and \( u \in s \) the concatenation \( w \) of \( t \) and \( u \) is a tuple over the relation schema \( R \cup S \) such that for each attribute \( A \in R \) then \( w[A] = t[A] \) and for each attribute \( B \in S \), \( w[B] = u[B] \). The concatenation is denoted by \( tu \). The Cartesian product \( r \times s \) of relations \( r \) and \( s \) is a relation over \( R \cup S \) consisting of all concatenations of tuples from \( r \) and tuples from \( s \) defined as

**Definition (CARTESIAN PRODUCT).**

\[ r \times s = \{ tu \mid t \in r \land u \in s \}. \]

If schemas \( R \) and \( S \) contain attributes with the same name, the Cartesian product is defined by first renaming an attribute \( B \) in \( R \setminus S \) as \( R.B \) in \( R \) and \( S.B \) in \( S \). Then the Cartesian product is a relation over the schema \( T = (R \setminus S) \cup (S \setminus R) \cup \{R.B, S.B \mid B \in R \cap S\} \).

The join operation may be used to combine related tuples from two relations. Two tuples are connected if the attributes that are common to both tuples have the same values. The join of \( r \) and \( s \) can be obtained from the Cartesian product of \( r \) and \( s \) by first selecting all the tuples with the same values for the attributes occurring in both \( R \) and \( S \). Then the duplicate occurrences of attributes are dropped using projection.

Formally, let

\[ r \bowtie s = \sigma_\phi(r \times s), \text{ where } \phi = \bigwedge_{B \in R \cap S} (R.B = S.B) \]

be the condition defining which tuples from the Cartesian product belong to the join. Then for each attribute \( B \in R \cap S \), attribute \( S.B \) must be removed and attribute \( R.B \) renamed back to \( B \). Given \( T \) where \( T = (R \setminus S) \cup (S \setminus R) \cup \{R.B, S.B \mid B \in R \cap S\} \) the join operation is defined as

**Definition (JOIN).**

\[ r \bowtie s = \rho_{R.B \rightarrow B \mid B \in R \cap S}(\pi_X(r \bowtie s)) \text{ where } X = T \setminus \{S.B \mid B \in R \cap S\} \]

A join operation may take two forms, a theta join (described above) denoted by \( \bowtie \), and a natural join denoted by \( \star \). The natural join may be viewed as the same as the theta join.
except that the join attributes of \( s \), are not included in the resulting relation. Furthermore if the join attributes have the same names, they do not have to be specified.

Let \( x \) denote a sequence containing the attributes in \( R \cap S \), then to aid clarity \( r \bowtie_x s \) will be written in place of \( r \bowtie s \) and \( r \star_x s \) will be written in place of \( r \star s \).

In the definitions of the operations PROJECTION, SELECTION and JOIN given above it is assumed that these operations are restricted so that they may not effect the ordering of the relations which are used as their parameters. For example the ordering of tuples in a relation \( r \), before a projection of that relation \( r \) on \( X \), is assumed to remain the same after the projection operation has been completed.

F.2.2 Update Operations

The two basic update operations on relations are \textit{Insert} and \textit{Delete}. The \textit{Insert} operation is used to insert tuples into a relation. The \textit{Delete} operation is used to delete tuples from a relation.

\textbf{Definition (Insert).} The \textit{Insert} operation is the union of a relation \( R \) with a set \( t \) of tuples.

The insertion of the tuple \( t \) into \( R \) to yeild a new relation instance \( R' \) can be written as follows.

\[ R' = R \cup t \]

Let \( t \) denote a tuple in \( R \), then \( R' = \text{Insert}(R,t) \) may be written in place of \( R' = R \cup t \).

\textbf{Definition (Delete).} The \textit{Delete} operation is the difference between a relation \( R \) and a set \( t \) of tuples.

The deletion of the tuple \( t \) from the relation \( R \) will be written to yeild a new relation instance \( R' \) can be written as follows.

\[ R' = R \setminus t \]

Let \( t \) denote a tuple in \( R \), then \( R' = \text{Delete}(R,t) \) may be written in place of \( R' = R \setminus t \).
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Adaptation</strong></td>
<td>(respectively, adaptivity) The process (respectively, the capability) of carrying out adaptive actions.</td>
</tr>
<tr>
<td><strong>Adaptive Action</strong></td>
<td>A system-initiated tailoring action.</td>
</tr>
<tr>
<td><strong>CAL</strong></td>
<td>Computer-based learning</td>
</tr>
<tr>
<td><strong>CBT</strong></td>
<td>Computer-based tutoring.</td>
</tr>
<tr>
<td><strong>CF</strong></td>
<td>Cold Fusion application server.</td>
</tr>
<tr>
<td><strong>CFML</strong></td>
<td>Cold Fusion mark-up language.</td>
</tr>
<tr>
<td><strong>DBS</strong></td>
<td>Database management system.</td>
</tr>
<tr>
<td><strong>DBS</strong></td>
<td>Database server.</td>
</tr>
<tr>
<td><strong>Goals-library</strong></td>
<td>A store of information goals.</td>
</tr>
<tr>
<td><strong>GUI</strong></td>
<td>A graphical or textual user interface.</td>
</tr>
<tr>
<td><strong>HBMS</strong></td>
<td>Hyperbase management system.</td>
</tr>
<tr>
<td><strong>HLBS</strong></td>
<td>Hyperlink-based system.</td>
</tr>
<tr>
<td><strong>HTML</strong></td>
<td>Hypertext mark-up language.</td>
</tr>
<tr>
<td><strong>Hyper-library</strong></td>
<td>A store of hyperpages and hyperpage annotations.</td>
</tr>
<tr>
<td><strong>Hyper-network</strong></td>
<td>A network of information units which may comprise text, graphics, video, animation and/or sound.</td>
</tr>
<tr>
<td><strong>Hyperdocument</strong></td>
<td>A collection of hyperpages.</td>
</tr>
<tr>
<td><strong>Hyperpage</strong></td>
<td>An information unit that can be rendered in a HLBS.</td>
</tr>
<tr>
<td><strong>Hyperpage specification</strong></td>
<td>A specification of how to build a hyperpage upon request.</td>
</tr>
<tr>
<td><strong>IMPS</strong></td>
<td>Intelligent multimedia presentation system.</td>
</tr>
<tr>
<td><strong>LLS</strong></td>
<td>Link server system.</td>
</tr>
<tr>
<td><strong>NCSA</strong></td>
<td>National center for supercomputing Applications.</td>
</tr>
<tr>
<td><strong>NLG</strong></td>
<td>Natural language generation.</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>OHLBS</td>
<td>Open hyperlink-based system.</td>
</tr>
<tr>
<td>P&amp;A</td>
<td>Personalisation and adaptation.</td>
</tr>
<tr>
<td>PA-HLBSs</td>
<td>Personalisable, adaptive hyperlink-based systems.</td>
</tr>
<tr>
<td>PAS</td>
<td>A personalisable, adaptive hyperlink-based system developed at Goldsmiths college.</td>
</tr>
<tr>
<td>Personalisation</td>
<td>(respectively, personalisability) The process (respectively, the capability) of carrying out personalisation actions.</td>
</tr>
<tr>
<td>Personalisation Action</td>
<td>A user-initiated tailoring action.</td>
</tr>
<tr>
<td>SMIL</td>
<td>The Synchronised Multimedia Integration Language.</td>
</tr>
<tr>
<td>Tailoring Action</td>
<td>An action that effects a transfer of ownership of a hyper-document from its designers to users.</td>
</tr>
<tr>
<td>UIS</td>
<td>User-interface server.</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator.</td>
</tr>
<tr>
<td>W3C</td>
<td>WWW Consortium.</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web.</td>
</tr>
<tr>
<td>XML</td>
<td>W3C extensible mark-up language.</td>
</tr>
</tbody>
</table>
Bibliography


329


