The Transaction Pattern through Automating TrAM

Ivan Launders[1], Simon Polovina[2], and Richard Hill[2]

[1] BT Global Services, PO Box 200, London, United Kingdom
ivan.launders@bt.com
[2] Cultural, Communication & Computing Research Centre (CCRC)
Sheffield Hallam University, Sheffield, United Kingdom
{r.hill,s.polovina}@shu.ac.uk

Abstract. Transaction Agent Modelling (TrAM) has demonstrated how the early requirements of complex enterprise systems can be captured and described in a lucid yet rigorous way. Using Geerts and McCarthy's REA (Resource-Events-Agents) model as its basis, the TrAM process manages to capture the 'qualitative' dimensions of business transactions and business processes. A key part of the process is automated model-checking, which CG has revealed to be beneficial in this regard. It enables models to retain the high-level business concepts yet providing a formal structure at that high-level that is lacking in Use Cases. Using a conceptual catalogue informed by transactions, we illustrate the automation of a transaction pattern from which further specialisations impart a tested specification for system implementation, which we envisage as a multi-agent system in order to reflect the dynamic world of business activity. It would furthermore be able to interoperate across business domains as they would share the generalised TM as a pattern.

1 Introduction

Transactions form a crucial part in enterprise systems and should therefore form a crucial part of their design. A central element of a transaction is the exchange of resources between agents. Transaction Agent Modelling (TrAM) exploits the formal underpinnings of Conceptual Graphs (CG) and Economic Accounting, a transactions-oriented architecture that is based upon Geerts and McCarthy's Resource-Events-Agents (REA) model [4],[12],[13],[15]. REA enables models to be built that reflect business activities which may include economic transactions. These models use the following core concepts:

- *Resource* - Any resource that is the subject of an exchange or *transaction*;
- *Event* - The activities that are required for a transaction to take place;
- *Agent* - A person, system or organisation that participates in the transaction.

TrAM, based on REA, captures the critical 'qualitative' dimensions of business transactions and business processes. These dimensions (e.g. 'quality of life') don't always lend themselves to be measured in monetary terms, but need to be

factored into design in order to usefully support business decisions. TrAM explicates and tests the subjective human judgement that otherwise leads to errors of omission or commission in capturing these qualitative nuances of transactions that nonetheless have significant consequences [13]. In outline, TrAM firstly captures the concept of a transaction, referred to as the Transaction Model (TM), permitting high-level models including their qualitative dimensions to be rapidly constructed and evaluated during the early requirements phase [7],[8]. This process ensures that vital domain knowledge is captured and retained from the outset so it is not lost in the later phases that lead to system prototyping and implementation. Secondly, it enables the validation of the generated models in a formal manner that is based on the TM, with business scenarios expressed as rules. This process specialises the TM according to each particular business domain, thus providing a transaction pattern from which these specialisations impart a tested specification for system implementation. We envisage this scenario as a multi-agent system in order to reflect the dynamic world of business activity. It would furthermore be able to interoperate across business domains as they would share the generalised TM as a pattern [13]. This is aided by using a generic ontology in the form of a 'conceptual catalogue', as explained later. The explication and testing of transactions is underpinned by the formal rigour underlying CG, including a range of computing operations that can be performed with them. Indeed, prior work has identified that CG software tools, namely Amine (http://amine-platform.sourceforge.net/), enable TrAM to be developed further by automating certain key CG operations such as projection, specialisation and maximal join [17]. Since it is a rich and well engineered application of CG theory in software, Amine also provides a model-checking environment that also assists the debugging of ontologies by identifying erroneous inconsistencies. TrAM is illustrated in Figure 1, which describes the overall stages in designing and implementing an enterprise system:

1. Initial concept analysis with UML use cases and CG;
2. Refine requirements analysis with CG and UML use cases;
3. Inference against the TM and verify;
4. Translate to an implementation design specification (e.g. UML);
5. Implementation (as a multi-agent system, MAS, or an alternative form of enterprise application).

Figure 1 however describes TrAM as a manually-oriented process, where the designer or knowledge engineer draws and evaluates diagrams (in this case CG) rather like an analyst uses UML diagrams (www.uml.org) to design a system. Whilst for UML there are numerous open-source or commercial tools to automate the analysis, design and implementation process (such as Together, www.borland.com/us/products/together/), for CG these remain underdeveloped. For TrAM to be sustained it needs to integrate a rich automation process in order to verify and test enterprise transaction models. TrAM has remained with CG rather than UML as our experiences have shown that CG focus on business rather than system models, as highlighted later. (A detailed evaluation is a current project, with the results expected to be published by 2010.) We hope

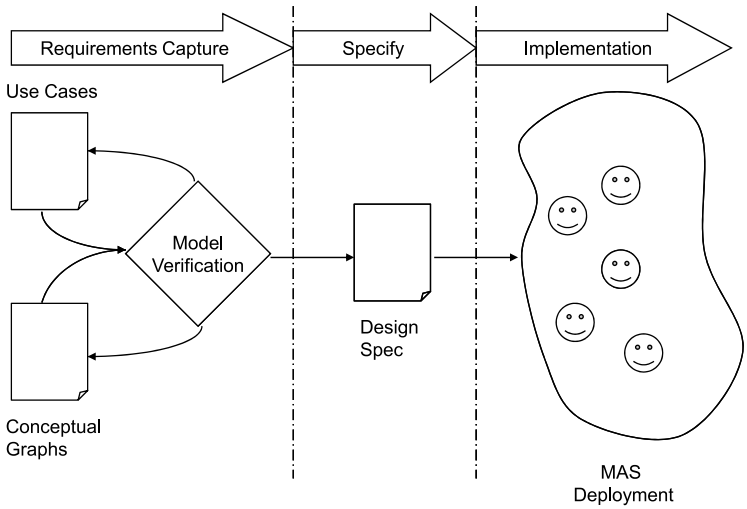

Fig. 1. The original TrAM framework.

that TrAM will encourage Amine and other CG tools out of the research labs into mainstream use. An overall approach for this automated form of TrAM is as follows:

1. Conduct Transactional Use-Case (TUC) Analysis;
2. Transform into CG;
3. Create and Refine the Type Hierarchy;
4. Build the TM Ontology;
5. Include the Conceptual Catalogue;
6. Refer to the Generic Transaction Model (TM);
7. Implement the TM Pattern;
8. Test and Refine the TM with Business Rules.

2 The TrAM Automation Process

This enhanced TrAM is illustrated by Figure 2, which also shows iteration across the domain ontologies and the conceptual catalogue. It exploits the productivity of computers rather than the manual human-based approach inherent to Figure 1. Figure 2 frees the human to focus on the creativity in conceptualising business transactions, leaving the mechanical checking and processing of the consequent transaction models to the computer. With this automated TrAM we accordingly have an integrated conceptual structure, where the human expert can articulate their knowledge through concepts as CG structures that by virtue of their operations as previously noted, can be computer processed. As the differences between Figure 1 and Figure 2 may be rather subtle, the actual process is explicated in detail as follows.

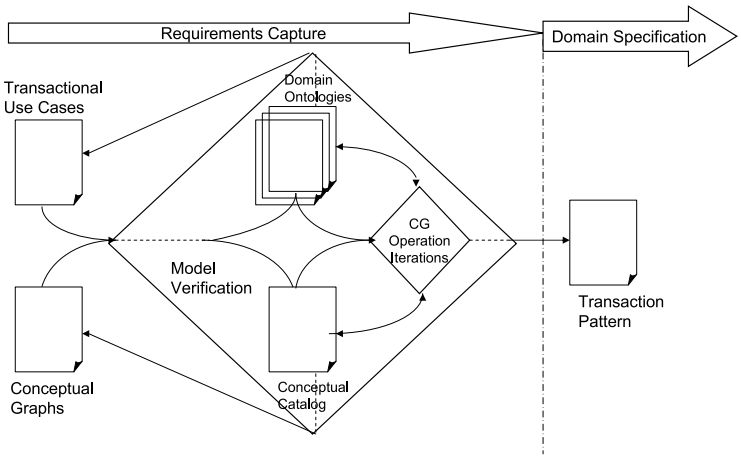

Fig. 2. TrAM, with automation.

2.1 Step 1 - Conduct Transaction Use-Case (TUC) Analysis

Transactional Use-Case (TUC) analysis provides a starting point to drive the process. The practice of use-case analysis defines a use case as *"a particular form or pattern or exemplar of usage, a scenario that begins with some user of the system initiating some transaction or sequence of interrelated events"* [10]. Use-case analysis is a manual process that enumerates the scenarios that are fundamental to the enterprise system transactions. To assist in this delineation, Fowler describes the distinction between low level system use cases and high level business use cases [2]. A system use case is the interaction with the software, whereas a business use case discusses how a business (enterprise) responds to a customer or event. TrAM focuses on business transactions and therefore is at business level. But what do we actually mean by a TUC? Let us explore this in a use-case diagram. We want to see the 'system' (the box) in the use-case diagram to represent the enterprise. As a transaction the activities of the UML actors are drawn in a way that they 'balance' each other out. Each actor is linked to a task that shows a process e.g. 'manage' associated with a resource e.g. 'care'. This maps to the TM as the former is an Economic Event whilst the latter is an Economic Resource. At this stage we begin to identify what are the inside and outside agents in our enterprise system and how they transact with each other through events and resources. Figure 3 is a simple TUC that illustrates a community healthcare domain example [14]. In this example there are three agents, where the inside agent (an elderly person) transacts with two outside agents, in this case to obtain the care she needs. It gives a high level view in terms of identifying events and balancing resources between agents that TrAM would then explicate.

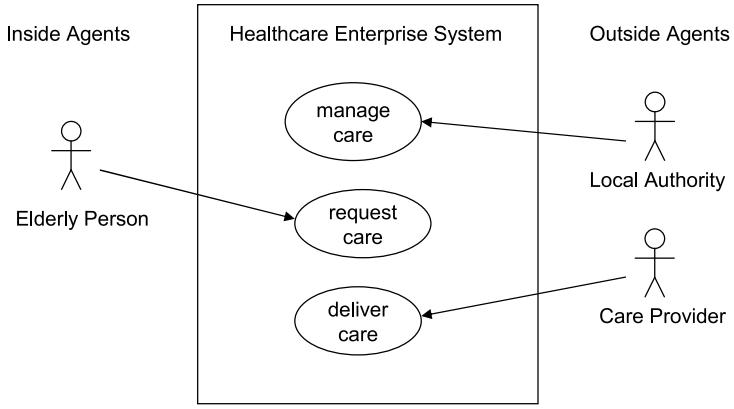

Fig. 3. Transactional Use-Case.

2.2 Step 2 - Transform into CG

The next step is to translate the Transactional Use-Case into CG. Figure 4 provides the CG for the community healthcare exemplar. In TrAM the initial analysis with CG and TM is an iterative step leading to a refined requirements analysis, progressively refining the easily readable but comparatively informal TUC model with the rigour and formality of CG operations such as specialisation. TUC analysis captures process-level tasks without unduly compromising

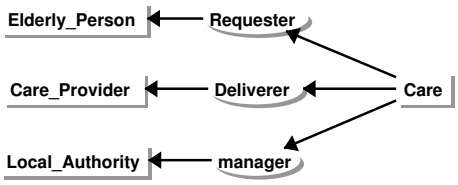

Fig. 4. The TUC in CG.

the transaction's qualitative dimensions. If the potential of an agent-based (or enterprise) system is to be realised, then the agents must be able to understand and process decisions or actions that require qualitative reasoning. Generation of terms for an ontology is largely based upon the existing processes together with the system analyst's knowledge and experience. From an enterprise system modelling perspective, the process of describing and articulating TUC serves to capture agent interactions with resources, and reactions to events as we have seen.

2.3 Step 3 - Create and Refine the Type Hierarchy

Figure 5 illustrates the initial type hierarchy for the healthcare scenario. It is the starting point for building the ontology for this domain. Applying the TrAM process involves expanding the type hierarchy. Refinement in the type hierarchy brings an alignment of each category in the TM domain ontology; refinement can also define a partial ordering of an ontology. A comprehensive type hierarchy also needs to capture and include relations. Types and relations are augmented by some generic ontology not specific to business transactions. It serves to reflect business diversity by allowing the TM for one domain to interoperate with TMs of other domains according to more generic terms. As well as enabling enterprises to discover hitherto unidentified business transactions that reflect the wider scope of their capabilities, it helps identify the TM as a design pattern [1], [3]. As indicated earlier, we refer to this generic ontology as a conceptual catalogue.

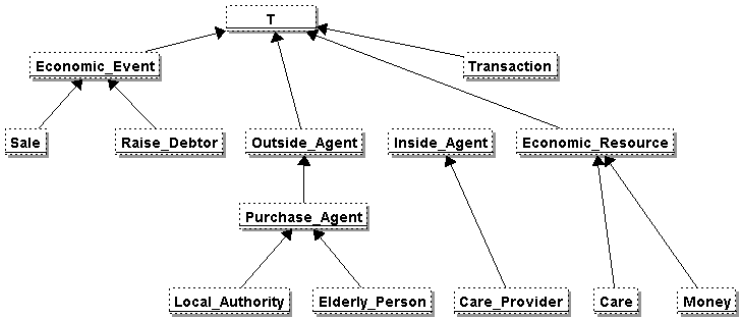

Fig. 5. Type Hierarchy for Health Care.

2.4 Step 4 - Build the TM Ontology

Gruber describes the meaning of ontology as a formal specification of the terms in a domain and the relationship between them [5]. An ontology defines a common vocabulary for agents who need to share information in a domain. It adds to the type hierarchy by containing the definitions of the basic concepts in the domain and the relations among them. Accordingly Figure 6 illustrates the healthcare ontology in Amine showing the implementation of its type hierarchy including relational types and subtypes for a general non-TM specific conceptual catalogue item 'source'. The relational CG for source is described through both a canon and a definition. Building an ontology using TrAM is as previously identified an iterative process which will drive out CG logic errors and test design assumptions captured through the transactional use-case analysis. This occurs as a result of breaking down the CG's into manageable steps prior to performing functions

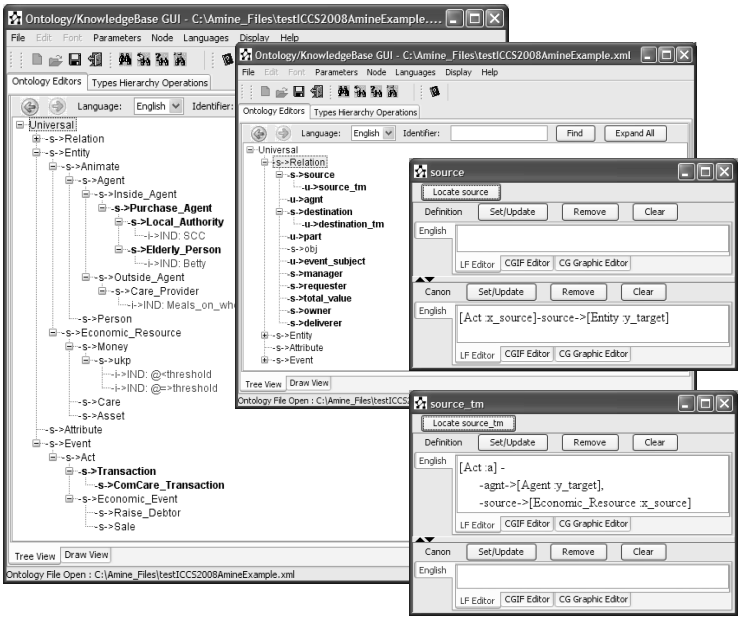

Fig. 6. Healthcare Ontology in Amine.

on them such as projection and maximal join. Amine's Ontology layer is used in order to create, edit, query, test and subsequently specialise the TM. The more specialised the TM the more it informs about what it is attempting to model. Using Amine's ontology layer we are therefore able to implement a refined type hierarchy, including relational types and subtypes. Building and testing a prototype TM ontology satisfies a key step in TrAM by verifying the conceptual analysis of an enterprise system.

2.5 Step 5 - Include the Conceptual Catalogue

Let us begin by explicating what is meant by the conceptual catalogue. A key element in the automation of TrAM is the inclusion of an ontology that is generic to both inside and outside business transactions. As our test we used Sowa's Conceptual Catalogue' (SCC) [17] (pp405-424). SCC provides a starter set, for an ontology with illustrative concepts and relations with their associated canons. Sowa subsequently developed SCC into a more comprehensive ontology [16]. SCC thus offers a simple but expressive vehicle for our comparative ontology. Figure 7 illustrates conceptual relations building up a conceptual catalogue for our example healthcare ontology. Source in the TM is identified as source_tm to distinguish it from source in Sowa's conceptual catalogue but they are nonetheless related. The canon for source is as defined in SCC, as it represents a canonical use of this term i.e. the concept types that 'source' relates to. In Figure 8 the definition for 'source_tm' is given. It arises from the TM rather than SCC. It

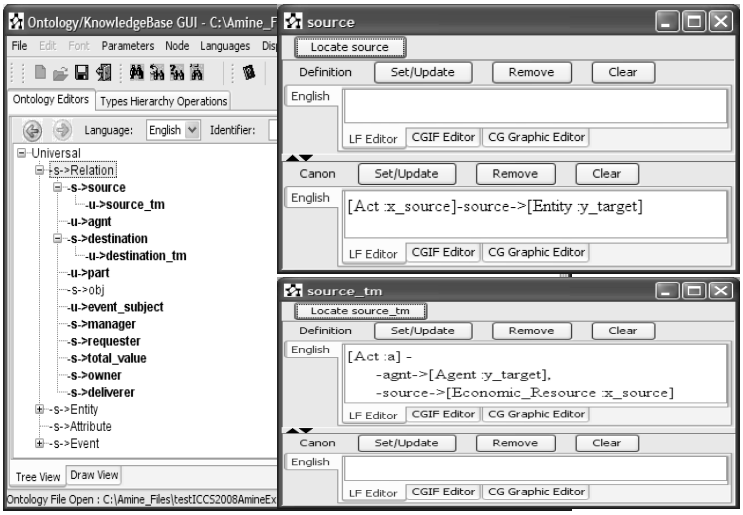

Fig. 7. Developing a TM Conceptual Catalogue in Amine.

is a definition because it is described in terms of its super-relation source i.e. source. It exemplifies how the TM is integrated with SCC. The other examples (e.g. 'destination') are given elsewhere [6].

2.6 Step 6 - Refer to the Generic Transaction Model (TM)

Core to TrAM is the TM. It enables models to retain the high-level business concepts yet it also provides a formal structure for the TUC. This is achieved in two ways. Firstly the concept of a transaction is captured by the TM, permitting high-level models including their qualitative dimensions to be rapidly constructed and evaluated during the early requirements phase. This process ensures that vital domain knowledge is captured and retained from the outset so it is not lost in the later phases that lead to system prototyping and implementation. Secondly, the TM enables us to validate the generated models in a formal manner that is based on the TM, with business scenarios expressed as rules as illustrated in Figure 10. These rules are used to improve the system specification in that they test the operation of the rule against the TM. This process specialises the TM according to each particular business domain. The TM thus stems from a generic TM that provides a transaction pattern across business domains as they would share this generalised TM as a pattern. Figure 8 illustrates the generic TM and is described in detail elsewhere [6-8, 13-15]. In summary the TM shows that that all transactions comprise of two economic events, denoted by *a and *b. The transaction is complete when both economic events balance, which indicates that *a always opposes *b, representing debits and credits. Additionally there are two related economic resources, *c and *d, each having independent source and destination agents and these are the agents

we have identified in our transactional use-case analysis. The Inside Agent and Outside Agent refer to the parties involved with the transaction. The Inside and Outside prefix denotes the relative perspective of the transaction for each party. The braces '*' denote plurality, indicating that each concept can represent a number of aggregated resources, events or agents. The TM is a formal structure of the TUC, embodying abstract concepts that can nonetheless be connected with SCC, specialised and tested across domains including by the automated processes we have described.

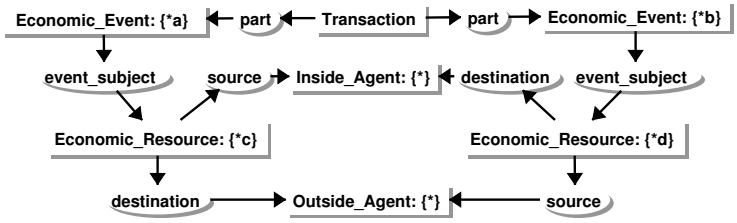

Fig. 8. The Transaction Model (TM).

2.7 Step 7 - Implement the TM Pattern

We have identified the TM new pattern in accordance with the expectations of software design and higher level patterns [1],[3]. Figure 9 illustrates the Amine implementation for the TM as a design pattern in CG. Amine enables the construction of an accurate implementation of the TM, developing in parallel a conceptual catalogue of conceptual relations.

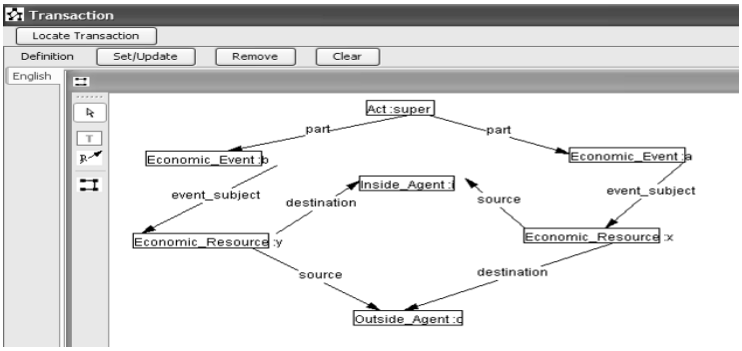

Fig. 9. Graphical View of TM in Amine.

2.8 Step 8 - Test and Refine the TM with Business Rules

Having built and verified the TM Ontology, the next part of the automation process is to use Amine's CGOperations interface to perform projection (subsume) and maximal join in order to implement business rules against the TM thus testing it and further specialising it. This is a stage that clearly benefits from automation. The human expert conceptualises these rules and the computer then runs the resultant structures. Previously in TrAM this was conducted manually, and certainly in our experiences most unsuccessfully. Undoubtedly therefore, automating this stage in TrAM provides the necessary refinements to the TM. It reveals design detail in type definition and canons providing further refinements. Figure 10 provides the CG for an example business rule in the community healthcare system. It shows that if an elderly person has assets below a certain threshold then the local authority is the destination (i.e. pays) for the care. This is described more precisely elsewhere [14]. The IF part (the antecedent) of the rule is projected (subsume in Amine) against a 'fact' CG that is provided and describes an actual business scenario. If it projects the THEN part of rule can undergo a Maximal Join to the TM, hence refining (specialising) it. Figure 11

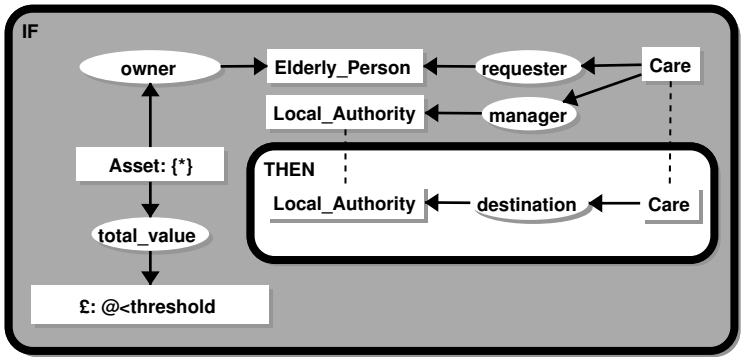

Fig. 10. Healthcare System Business Rule.

shows the Amine implementation for the fact to project the rule into (note there should be referents in the concepts to explicate it is a fact). In this case our example fact to test against (the projection) would be successful. Should it not project it implies that the TM cannot be refined according to this scenario; the TM may thus not reflect the enterprise or the business rule itself is inaccurate. As such it tests the business rules as well as the TM. Figure 12 shows the successful projection (subsume) operation in Amine. Input CG1 on the left hand side of the illustration contains the IF part. Input CG2 on the right hand side of the screen shot contains the fact to project into. Figure 13 shows the subsequent successful maximal join in operation through the TM. The TM has inferred the value 'Local_Authority:SCC' from the business rule. Maximal join facilitates inference

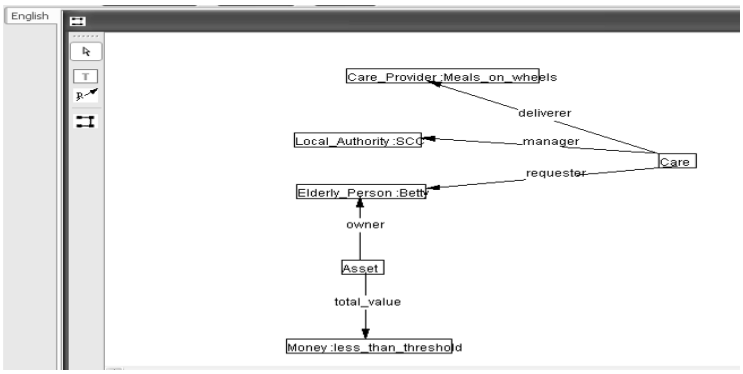

Fig. 11. Fact to Project Rule into, in Amine.

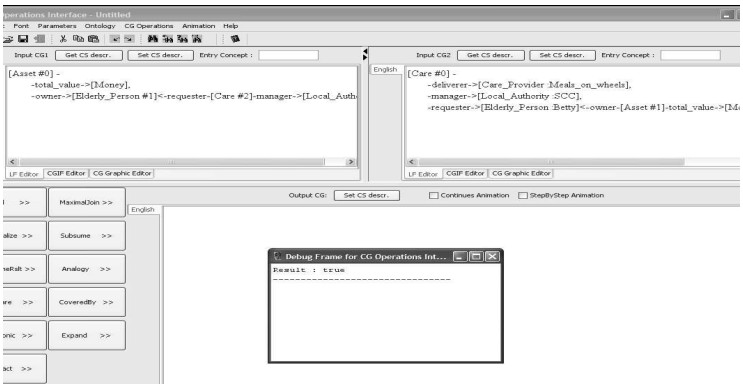

Fig. 12. Successful Projection of Rule into Fact.

because projections can be made into larger graphs containing more enterprise system information. The maximal join occurs on the maximally extended projection. In summary the rules part of the process is achieving inference in two

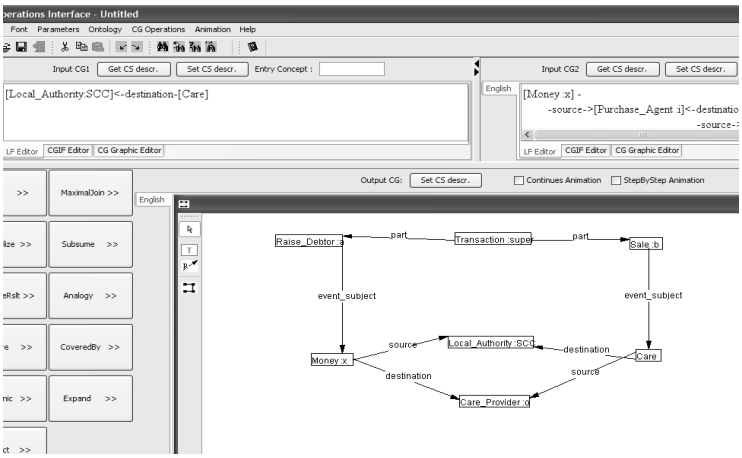

Fig. 13. Successful maximum join in operation through the TM.

steps, by projection of the IF part (antecedent) and by assertion of the THEN part (consequent).

3 Results

Exploring the Transaction Pattern through the automation of TrAM has firstly helped to clarify the process for a TM design pattern. A TM design pattern then becomes a general repeatable solution (a sequence of steps and re-usable CG's) to be stored and re-used. An implementation of the TM in Amine for the healthcare domain ontology has also revealed how beneficial it is to be able to build an automated model for TrAM. Implementing the TM pattern in this automated way significantly helps verify the TrAM framework in that it brought about refinement in the type hierarchy aligning categories in the domain ontology as well as developing a TM conceptual catalogue. The operational model quickly revealed errors and tested design assumptions revealing that an incomplete transactional-use case analysis and ultimately incomplete domain ontology will not give the desired results once business rules are applied. Ultimately the operational model as exemplified by the healthcare example showed how successful projection and maximal join operations were able to merge facts into the TM, adding to the qualitative outcomes. Figure 13 shows the result of the rules part of the TrAM process achieving inference (inferring the value 'Local_Authority:SCC' from a business rule) which typifies the effectiveness of automation.

4 Concluding Remarks

Exploring the Transaction Pattern through the automation of TrAM provides a sound starting point for the implementation of a transaction pattern as an architecture for implementing multi-agent enterprise systems. The benefit of model verification from the implementation of the TM domain ontology and a conceptual catalogue, now permits the designer to examine the transaction model in operation. Implementation or prototyping reveals design detail in terms of CG behaviour within the TM as well as testing the proof of business rules when working with the ontology. Amine allows the designer to verify the model, through the refinement of the TM ontology, editing, querying, testing and subsequently specialising the TM. Amine provides useful automation by checking type hierarchies; however it is not easy to adjust and edit the ontology in Amine if an error is made a in the upper levels of the ontology. Whilst such issues may typify the immaturity of CG tools in general, we are of the view that applications will drive the development of CG tools. TrAM, even with this automation, thus remains in development when compared to arguably, less expressive, but more integrated UML tools.

5 Acknowledgements

This work has been assisted by Amine's author, Adil Kabbaj. We also acknowledge the support of our colleague Lynne Dawson. We thank the many students whom we have taught TrAM to and reminded us of CG's general applicability in the real world. Part of this project was in receipt of an AgentCities Deployment Grant from the European Union AgentCities.rtd Project (IST-2000-28385).

References

1. de Moor, A. (2005) Patterns for the Pragmatic Web. In Proc. Of the 13th International Conference on Conceptual Structures (ICCS 2005), Kassel, Germany, July, 1-18,2005
2. Fowler. M. (2004) UML Distilled (Third Edition), Addison-Wesley, 103-104.
3. Gamma, E., Helm, R., Johnson, R., Vissides, J., (1994) Design patterns: Elements of reusable object-oriented software, Addison-Wesley.
4. Geerts, G. L., McCarthy. W. E. (1991). "Database Accounting Systems", in Information Technology Perspectives in Accounting: an Integrated Approach, Chapman and Hall, 159-183.
5. Gruber, T.R. (1993). A Transaction Approach to Portable Ontology Specification. Knowledge Acquisition 5: p199-220
6. Hill, R., Polovina, S., (2008) 'An Automated Conceptual Catalogue for the Enterprise', Supplementary Proceedings of 16th International Conference on Conceptual Structures (ICCS 09): Conceptual Structures: Knowledge Visualization and Reasoning, Toulouse, France, July 2008, Eklund, P., Haemmerl, O. (Eds.), CEUR-WS, Vol-354, 99-106.

7. Hill, R. (2006). 'Capturing and Specifying Multi-Agent Systems for the Management of Community Healthcare' in Yoshida, H., Jain, A., Ichalkaranje, A., Jain, L.C., Ichalkaranje, N., editors, 'Advanced Computational Intelligence Paradigms in Healthcare - 1', Chapter 6, 127-164, Studies in Computational Intelligence, 48, Springer, Berlin.
8. Hill, R., Polovina, S., Beer, M. D. (2005) "From concepts to agents: Towards a framework for multi-agent system modelling", Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Utrecht University, Netherlands, ACM Press, 1155-1156.
9. Hrub, P., Kiehn, J., & Scheller, C. V. (2006). Model-driven design using business patterns. Berlin; New York: Springer-Verlag.
10. Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G. (1992) Object-Oriented Software Engineering. Wokingham, England: Addison-Wesley.
11. McCarthy, W. E., (1982) "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment", The Accounting Review, 554-578.
12. McCarthy, W. E., (1979) "An Entity-Relationship View of Accounting Models", The Accounting Review, 667-686.
13. Polovina S., Hill, R. (2009) "A Transactions Pattern for Structuring Unstructured Corporate Information in Enterprise Applications", International Journal of Intelligent Information Technologies, April-June 2009, Vol. 5, No. 2, IGI Publishing, 34-48.
14. Polovina, S., Hill, R. (2005). "Enhancing the Initial Requirements Capture of Multi-Agent Systems through Conceptual Graph", Proceedings of 13th International Conference on Conceptual Structures (ICCS '05): Conceptual Structures: Common Semantics for Sharing Knowledge, July 18-22, 2005, Kassel, Germany, Dau, F., Mugnier, M-L., Stumme, G. (Eds.); LNAI 3596, Springer, 439-452.
15. Polovina, S. (1993) "Bridging Accounting and Business Strategic Planning Using Conceptual Graphs", Conceptual Structures: Theory and Implementation, Pfeiffer, Heather D; Nagle, T. (Eds.), LNAI, Springer-Verlag, Berlin, 312-321.
16. Sowa, J. F., (2000) Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA.
17. Sowa, J. F., (1984). Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley.