

The *Tropos* Metamodel and its Use

Angelo Susi and Anna Perini and John Mylopoulos
 ITC-irst, Via Sommarive, 18, I-38050 Trento-Povo, Italy
 susi@itc.it, perini@itc.it, jm@cs.toronto.edu

Paolo Giorgini
 Department of Information and Communication Technology
 University of Trento, via Sommarive 14, I-38050 Trento-Povo, Italy
 paolo.giorgini@dit.unitn.it

Keywords: Agent Oriented Software Engineering Methodology, Metamodel

Received: May 9, 2005

Tropos is a software development methodology founded on the key concepts of agent-oriented software development. Specifically, Tropos emphasizes concepts for modelling and analysis during the early requirements phase. This phase precedes the prescriptive requirements specification of the system-to-be. In this paper, we present the Tropos metamodel starting from the basic concepts of actor, goal, plan, resource and social dependency and then we illustrate its use by introducing an extension intended to introduce concepts for modelling security concerns. We also sketch the Tropos modelling environment and compare with the metamodels of other software development methodologies.

1 Introduction

Software development paradigms have exploited a wealth of models to capture requirements and design information about a software system (the “system-to-be”) throughout its development process. Structured software development used SADT and Data Flow Diagrams. Object-oriented software development has used a range of modelling languages which have been integrated into UML. Not surprisingly, agent-oriented software development is following on the same footsteps.

To formally analyze software models, we need a means to define their syntax and semantics. *Metamodels* have been used for the former task. Metamodels define a set of possible instantiations, which are all and only the syntactically correct models in some modelling language. As such, metamodels have been used for more than two decades as a basis for defining the syntax of (usually graph-theoretic) modelling languages, such as UML as well as *Tropos*.

The objective of this paper is to introduce the *Tropos* metamodel, discuss some of its uses, and

compare it to other metamodels of agent/goal-oriented software development methodologies. Section 2 of the paper sketches the *Tropos* methodology, while Section 3 presents the metamodel and explains its features. Section 4 presents one extension of the metamodel to include security-related concepts. In Section 5 we sketch the *Tropos* development environment, which uses the metamodel in its basic core. Section 6 relates the proposed metamodel to others in the same family of modelling languages, while Section 7 concludes the paper.

2 Models and Methodology

Tropos is founded on the idea of using the agent paradigm and related mentalistic notions during all phases of the development software process. The methodology [6] adopts the *i** [26] modelling framework, which proposes the concepts of (social) *actor*, *goal*, *task*, *resource* and *social dependency* to model both the system-to-be and its organizational operating environment. The *i** framework includes the strategic dependency

model (actor diagrams in *Tropos*) for describing the network of inter-dependencies among actors, as well as the strategic rationale model (goal diagrams in *Tropos*) for describing and supporting the means-ends analysis conducted by each actor as it attempts to ensure that – through delegations to other actors – its goals will eventually be fulfilled.

An *actor diagram* is a graph whose nodes represent actors (*agents*, *positions*, or *roles*), while edges represent dependencies among them. A dependency represents an agreement between two actors where one actor (the depender) depends on another (the dependee) to fulfill a goal, perform a task or deliver a resource (the dependum). Dependencies may also involve softgoals (such as “having a good quality meeting”) which represent vaguely-defined goals, with no clear-cut criteria for their fulfillment.

A *goal diagram* is also a graph where nodes represent goals or plans¹, while edges represent goal/plan relationships, such as AND/OR-decomposition (i.e., a goal/plan can be decomposed into a set of other goals/plans. Goals/plans can also be related to softgoals through qualitative relationships (labelled “+” or “-”) to indicate that the goal/plan contributes positively or negatively to the fulfillment of the softgoal. Goal diagrams appear inside a balloon associated with a single actor. This is the actor whose goals/plans are being analyzed to determine how they can be fulfilled/executed.

The *Tropos* methodology supports four phases of software development: Early Requirements Analysis, Late Requirements Analysis, Architectural Design, and Detailed Design. *Early requirements* is concerned with understanding the organizational context within which the system-to-be will eventually function. During early requirements analysis, the requirements engineer identifies the domain stakeholders (who have a stake in the system-to-be) and models them as social actors, who have goals and depend on each other for goals to be fulfilled, plans to be performed, and resources to be furnished. *Late requirements*, on the other hand, is concerned with a definition of the functional and non-functional requirements of the system-to-be. This is accomplished by treating the system as another actor (or a small number

of actors) who are dependers/dependees in dependencies that relate them to external actors. The shift from early to late requirements occurs when the system actor is introduced and it participates in delegations from/to other actors.

Architectural design is concerned with the global structure of the system-to-be. Unsurprisingly, subsystems and system components are represented as actors too, and their dependencies to other system components are social, rather than procedural/structural. This means that system components need to have the ability to monitor dependencies to other actors to make sure they will be fulfilled. As well, system components need to be able to cancel dependencies that seem ineffective and replace them with new ones through planning, negotiation, etc. As with conventional software architectures, architectural styles constitute critical support for the software developer. Since the fundamental concepts of *Tropos* architectures are intentional and social, we have turned to theories which study social structures to define architectural styles: namely Organization Theory and Strategic Alliances.

Detailed design focuses on the specification of actor communication and behavior. To support this phase, we have adopted existing agent communication languages such as FIPA-ACL [20] or KQML [11]; also message transportation mechanisms and other related concepts and tools. We have also proposed and defined a set of stereotypes, tagged values, and constraints to accommodate *Tropos* concepts within UML [5].

Through the models constructed during these phases, one can answer “why” questions, in addition to “what” and “how” ones, regarding system functionality. For example, one can ask “Why does this component of the system need to notify library users when a book becomes available?”. Answers to why questions ultimately link system functionality to stakeholder needs, preferences and objectives. Such answers serve as ultimate justifications for all elements of a proposed design.

3 The Metamodel

Figure 1 shows the portion of the *Tropos* metamodel, where agent, role and position are specialization of the concept of actor. A position

¹Plans in *Tropos* correspond to tasks in *i**.

erationalized as send papers by e-mail, and select reviewers decomposed in verify the competences and verify conflicts. This latter represents the “Why” for the resource dependency conflicts between the PC Member and the reviewer. Moreover, the fulfillment of these two sub-goals can contribute positively to the fulfillment of the softgoal be fair in the review assignment as described by the positive contribution relationships in the diagram.

4 Metamodel Extension

Secure *Tropos* has been proposed in [16] as a formal framework for modelling and analyzing security. It enhances *Tropos* introducing four new concepts and relationships behind *Tropos* dependency: *trust*, *delegation*, *provisioning*, and *ownership*. The basic idea of ownership is that the owner of a resource (goal or plan) has full authority concerning access and disposition of his resource (goal or plan). The distinction between owning a resource makes it clear how to model situations in which, for example, a client is the legitimate owner of his/her personal data and a Web Service provider that stores customers’ personal data, provides the access to her/his data. We use the relation for delegation when in the domain of analysis there is a formal passage of authority (e.g. a signed piece of paper, a digital credential is sent, etc.). The trust relations have their intuitive meaning among agents, namely the believe of an agent that the actor does not misuse some resources.

Figure 5 shows the the new part of the *Tropos* metamodel concerning trust and ownership. An actor (the *truster*) trusts another actor (the *trustee*) about the achievement of a goal, the fulfillment of a plan or the delivering of a resource. The content of the trust relationship is called *trustum*. An actor can be the owner of a resource, a plan and goal and he/she has authority concerning the use of the resource, the execution of the plan and achievement of the goal, respectively.

The metamodel describing delegation relationships is basically identical to the metamodel for the dependency relationship as presented in Figure 1. The *delegater* delegates the *delegatee* for the achievement of a goal, the execution of a plan or the delivering of a resource. As for the de-

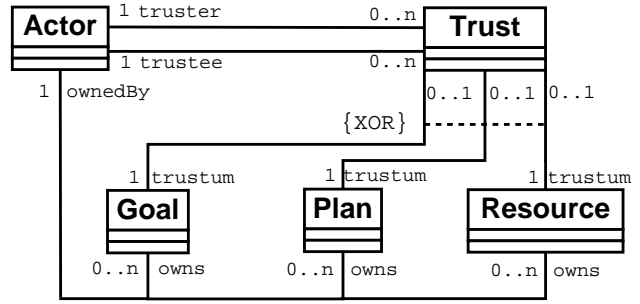


Figure 5: The *Tropos* metamodel related to the concept of Trust.

pendency relationship, it is also possible here to specify the reason (*why*) of a delegation.

We have shown in [17] how the original concept of *Tropos* dependency can be expressed in terms of trust and delegation. Roughly, when an actor depends on another actor to achieve a goal (to fulfill a task or to deliver a resource), it is implicitly intended that the actor trusts the other actor and delegates it for such activities. A precise formalization of dependency refinement in terms of trust and delegation has been presented in [17].

Figure 6 presents an example of application the extended metamodel. The Author trusts the PC Chair to implement a fair review process and he/she is the owner of the paper sent to the PC Member and reviewed by the Reviewer. The PC Chair trusts and delegates PC Member to review a certain number of papers, and in turn the PC Member trusts and delegates the Reviewer to review the papers. The PC member (Reviewer) depends on the PC Chair (PC Member) to receive the paper to review.

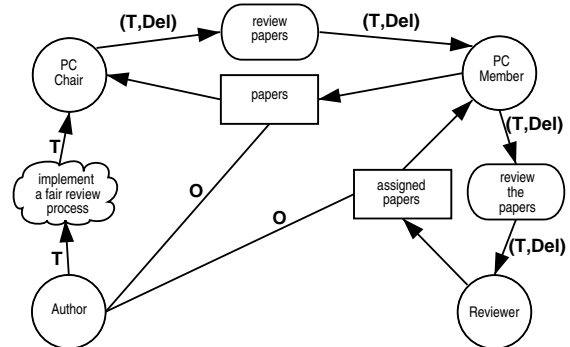


Figure 6: The *Tropos* actor diagram with the trust concepts.

5 A Modelling Environment

In order to support the specific analysis techniques adopted in *Tropos*, different tools have been developed, such as a tool for the verification of requirements specification through model-checking technique (T-Tool) [13], a tool which supports forward and backward reasoning on the goal analysis structures (GR-Tool) [15]. In this section, we will give details of a modelling environment, called TAOM4e (Tool for Agent-Oriented Modelling for Eclipse), which is based on an implementation of the metamodel described in the previous sections. The metamodel has been specified following the OMG’s MDA [21] standard for metamodel interoperability, that is the Meta Object Facility (MOF)² which offers a mechanism for automatically deriving a concrete syntax based on XML DTDs and/or schemas known as XML Model Interchange (XMI). This is a preliminary step towards the adoption of the model-to-model transformation approach proposed by MDA.

Among the main requirements we considered in developing this tool are the following [23]:

- *Visual Modelling.* The modelling environment should support the user during the specification of an AO model (e.g., according to the *Tropos* visual notation). Moreover, the environment should allow us to represent new entities that will be included in the *Tropos* metamodel, language variants, such as those presented in Section 4, as well as to restrict its use to a subset of entities of the modelling language.
- *Specification of model entities properties.* The modelling environment should allow us to easily annotate the visual model with model properties like invariants, creation or fulfillment conditions that are typically used in Formal *Tropos* specification.
- *Automatic Model Translation.* The modelling environment should allow us to save a model in a standard format (e.g., XML and XMI), and provide automatic transformation into a different specification language. The model-to-model transforma-

tion approach should be also compliant with Query/View/Transformation (QVT) requirements [14], as discussed in [24].

- *Extensibility.* The modelling environment should be extensible and allow for different configurations by easily integrating other tools at will.

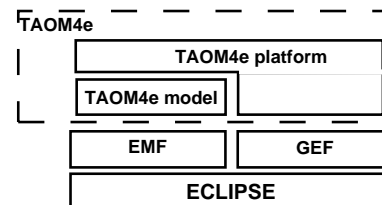


Figure 7: The architecture of TAOM4e.

An effective solution to the requirement of a flexible architecture and to the component integration issue is offered by the Eclipse Platform.

New tools are integrated into the platform through plug-ins that provide the environment with new functionalities. A plug-in is the smallest unit of function in Eclipse and the Eclipse Platform itself is organized as a set of subsystems, implemented in one or more plug-ins, built on the top of a small runtime engine. The TAOM4e architecture is depicted in Figure 7. It follows the Model View Controller pattern and has been devised as an extension of two existing plug-ins. First, the EMF plug-in³ offers a modelling framework and code generation facilities for building tools and other applications based on a structured data model. Given an XMI model specification, EMF provides functions and runtime support to produce a set of Java classes for the model. Most importantly, EMF provides the foundation for interoperability with other EMF-based tools and applications. The resulting plug-in, called *TAOM4e model* implements the *Tropos* metamodel. It represents the Model component of the MVC architecture. Second, the Graphical Editing Framework (GEF) plug-in⁴ allows developers to create a rich graphical editor around an existing metamodel. The functionality of the GEF plug-in helps to cover the essential requirement of the tool, that is supporting a visual development of *Tropos* models by providing some stan-

²http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF

³<http://www.eclipse.org/emf/>

⁴<http://www.eclipse.org/gef/>

standard functions like drag & drop, undo-redo, copy & paste and others. The resulting plug-in, called *TAOM4e platform* represents both the Controller and the Viewer components of the tool. In Fig-

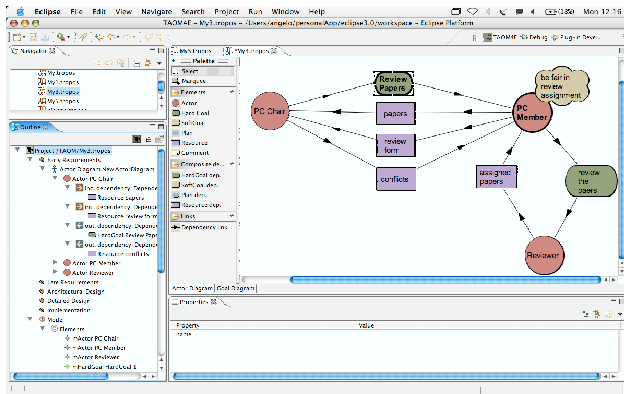


Figure 8: The Graphic User Interface of TAOM4e.

ure 8 a snapshot of the modeler: the diagram editor window on the right, the project and model browsers on the left, the entity properties window at the bottom.

6 Related Work

Many Agent-Oriented Software Engineering methodologies have been proposed and compared over the last few years [18, 25]. An analysis of the metamodels of three methodologies, ADELFE [4], GAIA [27] and PASSI [7] has been presented in [3]. The aim of this work was to face interoperability issues between different methodologies.

In this section we extend this analysis including *Tropos*. We will focus on four dimensions: Agent Structure, Agent Interaction, Agent Organization and Agent Development (e.g., CASE tools at support of the development process). Table 1 summarizes the comparison. In ADELFE the concept of agent (*Cooperative Agent*) is defined as the composition of aptitudes, skills, characteristic, communication and representation. Not explicit concept of role is given, the concept of goal is implicitly used to identify agent skills, but it is not representable as well as the concept of plan, since a plan is an entity that will be built at run time and which is not representable at design time. In GAIA, an agent (*Agent Type*) is specified as a composition of roles. Each role is

responsible of a specific set of activities associated with the role. Goals cannot be explicitly modeled, but they are implicitly used to characterize a role. In PASSI, an agent (*Agent*) is defined as the composition of roles and each role is defined as the manifestation of the agent activity in some scenario. Goals are implicitly considered when specifying non-functional requirements attached to agent duties. In *Tropos*, the concept of *Actor* generalizes the concepts of agent and role (or set of roles), an actor can have individual goals and it can be able to execute plans to satisfy goals. Goal analysis in *Tropos* drives the modelling process, as discussed in Section 2 and allows us to represent goal decomposition, means to satisfy a goal or contribution towards goal satisfaction through different goal relationships.

The concepts used to specify the interactions of an agent with another agent or with the environment are similar in ADELFE, GAIA and PASSI. Basically, they use the concept of communication, role, and protocols. *Tropos* adopts the Agent Unified modelling Language (AUML) Agent Interaction Diagram, described in [2, 22] (proposed by the FIPA –Foundation for Physical Intelligent Agents– [12] and the OMG Agent Work group) where agent communicative acts are represented as messages in a UML sequence diagram.

In GAIA, the concept of organization is a primary concept, organization rules specify constraints that the organization should observe. In PASSI, agent organization aspects are modeled implicitly in terms of services that can be accessed by agents in a given scenario. In ADELFE, agent organization and society emerges from the evolving interactions between the agents which are compliant with cooperation rules.

In *Tropos* the strategic dependencies between actors in a domain makes explicit the organizational dimension and provide basic entities to model organizational patterns [19]. Moreover, the *Tropos* metamodel has been extended to include concepts of business processes and security.

Both ADELFE and PASSI provide CASE tools at support of modelling and for ad-hoc analysis on part of the resulting specification. *Tropos* provides modelling and analysis tools (details can be found in <http://www.troposproject.org>) as well as code generation tools [10].

This comparison shows that different metamodel-

Agent Structure	ADELFE	GAIA	PASSI	Tropos
<i>Agent</i>	Cooperative Agent	Agent Type	Agent	Actor
<i>Role</i>	Not explicit	Role in a organization	Role in a scenario	Specialization of Actor
<i>Goal</i>	Not explicit	Not explicit	Not explicit	goal and goal relationships
<i>Plan</i>	Not explicit	Activity of a Role	Ontology of Action	Plan and plan relationships
Agent Interaction				
<i>Comm. & Protocol</i>	Agent Communication Agent Interaction Protocols associated to communication	Communication associated to a role and protocols associated to a communication	Communication associated to a role and Messages as components of communication	Not in the current metamodel. AUML interaction diagram UML sequence diagram messages for communication acts
A. Organization				
<i>Structure & Rules</i>	Cooperation rules	OrganizationStructure, Organization, OrganizationalRule	Not explicit	Strategic Dependency, Ownership, Delegation and Trust Organizational patterns
A. Development				
<i>Modeler</i>	Open-Tool	—	PASSI Toolkit	TAOM, OME, DW-Tool, ST-Tool
<i>Analysis tooos</i>	Open-Tool	—	PASSI Toolkit	GR-Tool, DW-Tool, ST-Tool, T-Tool
<i>Code Generation</i>	—	—	PASSI Toolkit	SKwYRL

Table 1: Comparison of the meta-models of four Agent-Oriented methodologies.

els (methodologies) may allow us to model different properties of a system (e.g., organizational aspects, communications and protocols). On the other hand, it shows that even if metamodels share a comparable set of concepts, they can be used in a different way by the different methodologies. This can be found also considering requirements engineering methodologies based on metamodels. For instance, in KAOS [9], the concept of agent is used to assign leaf goals resulting from goal analysis.

Finally, other related work on i^* and Tropos metamodels are worth to be mentioned. The i^* metamodel [26] represents the basis for the Tropos metamodel. Other extensions of the i^* metamodel have been proposed. For instance, in [1] where a methodology for COTS selection is proposed.

7 Conclusion

We have presented an overview of the Tropos metamodel. Like other software development methodologies, Tropos supports a variety of models that need to be analyzed for syntactic and semantic consistency. The metamodel serves as a basis for checking for syntactic consistency. Making it richer, could also help in supporting some forms of semantic consistency currently conducted through a series of tools offered within the Tropos software development environment.

References

- [1] C. Ayala, C. Cares, J. P. Carvallo, G. Grau, M. Haya, X. Franch G. Salazar, E. Mayol, and C. Quer. A Comparative Analysis of i^* -Based Agent-Oriented Modeling Language. In *Proc. of AOSDM Ws in SEKE'05 Conference*, July 2005.
- [2] B. Bauer, J.P. Muller, and J. Odell. Agent UML: A formalism for specifying multiagent interaction. In *Proc. of the 1st Int. Workshop on Agent-Oriented Software Engineering, AOSE'00*, pages 91–104, Limerick, Ireland, 2001.
- [3] C. Bernon, M. Cossentino, M. P. Gleizes, P. Turci, and F. Zambonelli. A Study of Some Multi-agent Meta-models. In *Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004*, volume 3382 of *LNCS*, pages 62 – 77, New York, USA, NY, July 2004.
- [4] C. Bernon, M.P. Gleizes, S. Peyruqueou, and G. Picard. ADELFE, a Methodology for Adaptive Multi-Agent Systems Engineering. In *Third International Workshop Engineering Societies in the Agents World (ESAW-2002)*, Madrid, Spain, 2002.
- [5] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language: User Guide*. Addison-Wesley, 1999.
- [6] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An Agent-Oriented Software Development

- Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, July 2004.
- [7] A. Chella, M. Cossentino, and L. Sabatucci. Tools and patterns in designing multi-agent systems with PASSI. *WSEAS Transactions on Communications*, 3(1):352 – 358, Jan 2004.
- [8] L. K. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.
- [9] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.
- [10] T. T. Do, M. Kolp, and S. Faulkner. Agent Oriented Design Patterns: The SKwyRL Perspective. In *Proc. of the 6th International Conference on Enterprise Information Systems (ICEIS 2004)*, Porto, Portugal, April 2004.
- [11] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J.M. Bradshaw, editor, *Software Agents*. MIT Press, 1997.
- [12] FIPA. The Foundation for Intelligent Physical Agents. At <http://www.fipa.org>, 2001.
- [13] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. Model checking early requirements specification in Tropos. In *Proc. of the 5th IEEE International Symposium on Requirements Engineering*, Toronto, CA, August 2001.
- [14] T. Gardner, C. Griffin, J. Koehler, and R. Hauser. A review of omg mof 2.0 query / views / transformations submissions and recommendations towards the final standard. In *MetaModelling for MDA Workshop*, York, England, 2003.
- [15] P. Giorgini, J. Mylopoulos, and R. Sebastiani. Goal-Oriented Requirements Analysis and Reasoning in the Tropos Methodology. *Engineering Applications of Artificial Intelligence*, 18(2), 2005.
- [16] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Requirements Engineering meets Trust Management: Model, Methodology, and Reasoning. In *Proc. of iTrust'04, LNCS 2995*, pages 176–190. Springer-Verlag, 2004.
- [17] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling Security Requirements Through Ownership, Permission and Delegation. In *Proc. of the The 13th IEEE Requirements Engineering Conference (RE'05)*, August-September 2005.
- [18] B. Henderson-Sellers and P. Giorgini, editors. *Agent-Oriented Methodologies*. Idea Group Inc., Hershey, PA, USA, 2005.
- [19] M. Kolp, P. Giorgini, and J. Mylopoulos. A goal-based organizational perspective on multi-agents architectures. In *Proc. of the 8th Int. Workshop on Intelligent Agents: Agent Theories, Architectures, and Languages, ATAL'01*, Seattle, USA, August 2001.
- [20] Y. Labrou, T. Finin, and Y. Peng. The current landscape of agent communication languages. *Intelligent Systems*, 14(2):45–52, 1999.
- [21] Stephen J. Mellor, Kendall Scott, Axel Uhl, and Dirk Weise. *MDA Distilled*. Addison-Wesley, 2004.
- [22] J. Odell, H. Van Dyke Parunak, and B. Bauer. Extending UML for agents. In *Proc. of the 2nd Int. Bi-Conference Workshop on Agent-Oriented Information Systems, AOIS'00*, pages 3–17, Austin, USA, July 2000.
- [23] A. Perini and A. Susi. Developing Tools for Agent-Oriented Visual Modeling. In G. Lindemann, J. Denzinger, I.J. Timm, and R. Unland, editors, *Multiagent System Technologies, Proc. of the Second German Conference, MATES 2004*, number 3187 in LNAI, pages 169–182. Springer-Verlag, 2004.
- [24] A. Perini and A. Susi. Automating Model Transformations in Agent-Oriented modelling. In *Agent-Oriented Software Engineer-*

- ing VI: AOSE 2005*, LNCS. Springer, August 2005.
- [25] A. Sturm and O. Shehory. A Framework for Evaluating Agent-Oriented Methodologies. In M. Winikoff P. Giorgini, B. Henderson-Sellers, editor, *Proc. of AOIS 2003*, number 3030 in LNCS, pages 94 – 109. Springer-Verlag, 2003.
- [26] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, 1995.
- [27] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317 – 370, July 2003.