

# Refining Prometheus Methodology with i\*

Gilberto Cysneiros

Andrea Zisman

Software Engineering Group,  
Department of Computing, City University  
Northampton Square, EC1V OHB, UK  
{g.cysneiros,a.zisman}@soi.city.ac.uk

**Abstract.** In the last years agent oriented paradigm has been extensively used to support complex, distributed, open, dynamic, unpredictable, heterogeneous, and highly interactive software system applications. One of the main problems to the success of agent oriented approaches in industrial settings is the lack of mature software development methodologies to assist with the whole life cycle of software system development. Prometheus methodology has been developed to overcome the problem above. However, Prometheus does not offer fully support for early requirements specification. In this paper we present an approach to enhance Prometheus by using i\* technique to allow both goals and business modelling. We describe some guidelines of how to generate Prometheus artefacts from i\*. Our approach is illustrated through a BookStore example.

## 1 Introduction

Current software system applications need to be deployed in complex, distributed, large, open, dynamic, unpredictable, heterogeneous, and highly interactive environments [26]. Moreover, according to Wooldridge [27], the history of computing has been marked by five trends, namely ubiquity, interconnection, intelligence, delegation, and human orientation. In order to develop and deploy software systems in the types of environments listed above, a new software engineering concept has emerged: the *agent oriented paradigm*.

Agent oriented paradigm addresses a number of problems that the traditional software engineering area can not do properly. Examples of these problems include, but are not limited to, the need for software systems to (a) exhibit rational and human-like behaviour, (b) support flexible patterns of interaction, and (c) provide better support for robustness. Agent oriented paradigm has demonstrated success in many application areas, such as telecommunications, manufacturing, finance, air traffic management, aerospace, e-commerce, customer management, military simulation, decision support, and games. An overview of agent applications is presented in [16]. However, in order to the agent oriented paradigm gain widespread acceptance in industrial and commercial scenarios it is necessary to have mature software development methodologies to support the development of such applications systems.

In the last years some methodologies for agent oriented systems have been proposed. Examples of some of these methodologies are Tropos [4, 14], Gaia [28], MaSE [11], and Prometheus [20]. However, to the best of our knowledge, none of these methodologies provides detailed support for the whole life-cycle of software system development. For instance, Tropos provides very good support for the early phase of software system development (early requirement analysis and requirement analysis) [25], but relies on UML [19] and AUML [18] as languages to represent the design phase. In addition, as mentioned in [9], Tropos does not provide complete support for the implementation phase since it lacks of detailed process, heuristics, and examples. Gaia provides extensive support for the analysis and design phases, but lacks of support for the other development phases. MaSE offers support for all phases but like in the others approaches the transition from design to implementation is not direct. Moreover, as stated in [25][9], Prometheus and Tropos support the use of mental attitudes such as belief, desires, and intentions, as opposed to MaSE that does not use these concepts throughout the whole development life-cycle.

Prometheus appears to be the most complete and mature of these methodologies since it provides support for system specification, architectural and detailed design, and implementation, and has been applied in both industrial and academic environments. However, Prometheus does not offer proper support for early requirements phase. It uses a simple version of KAOS [10] to describe the goals of the system, but description of business models is not supported.

Several techniques have been proposed to capture early requirements. In [12], the authors divide the work in the area in two streams: *goal modelling* and *business modelling*. Goal modelling techniques [1, 3, 10] use goals to represent the requirements of the system which are further mapped into use case models or services, but do not provide mechanisms to represent the structure of the business. As a consequence, these techniques do not allow support for other types of analyses such as business process reengineering analysis, dependency analysis, and workflow analysis that are important for the understanding of the organization by software developers. Business modelling techniques [6, 13, 29] have been proposed to address the above issues. They can be used to specify models of goals, structures, and processes of organizations. Such models can be applied to assist stakeholders and developers to have a common understanding of the organisation and, therefore, facilitate the identification of requirements of systems that are developed to support the organisation's functions. As suggested in [13], if the requirement specification of a software system that is to be deployed in an organisation is based on a model of this organisation, then it is possible to guarantee that the system will support the business in a more appropriate way.

In this paper we present an approach to enhance Prometheus methodology by allowing support for the early requirements phase for both goals and business modelling. We propose to use *i\** [29] technique since it has been recognized as one of the most important requirement engineering technique to represent business model. *i\** is a simple technique which contains all necessary elements to represent organisational models. Our work is based in Prometheus due to its large use in both industrial and academic settings, detailed support for most of the software

engineering development phases, and existence of tools to assist with its use (e.g. Prometheus Design Tool, auml, and Jack Development Environment).

In this paper we present relationships between i\* concepts and Prometheus documents used in the system specification phase. In particular, we present some guidelines of how to generate Prometheus use case scenarios and descriptors for functionality, actions, and percepts from i\* concepts.

The remaining of this paper is structured as follows. Section 2 gives an overview of the i\* technique and Prometheus methodology illustrated by an example of a bookstore that we will use throughout the paper to demonstrate our work. Section 3 discusses how i\* can be used to enhance Prometheus and presents guidelines of how to use the business model created in i\* to generate Prometheus system specification phase. Section 4 discusses related work. Finally, section 5 summarises our work and suggests directions for future work.

## 2 Overview of i\* Technique and Prometheus Methodology

In this section we present a brief description of i\* and Prometheus.

### 2.1 i\* Technique

The i\* technique provides mechanism to describe the organization in terms of intentional and strategic actors and their relationships. The framework is composed of two models: *Strategic Dependency (SD)* model and *Strategic Rationale (SR)* model.

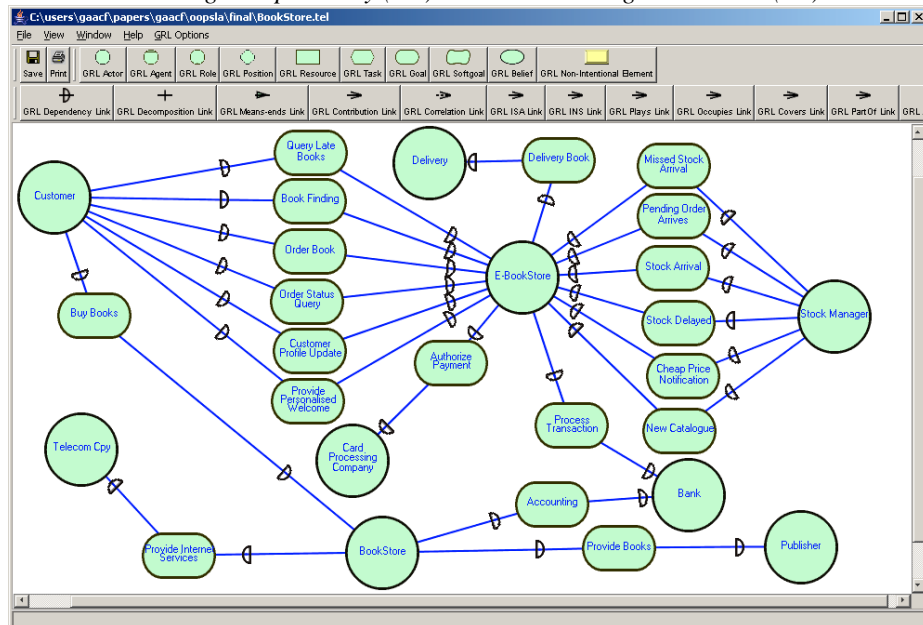
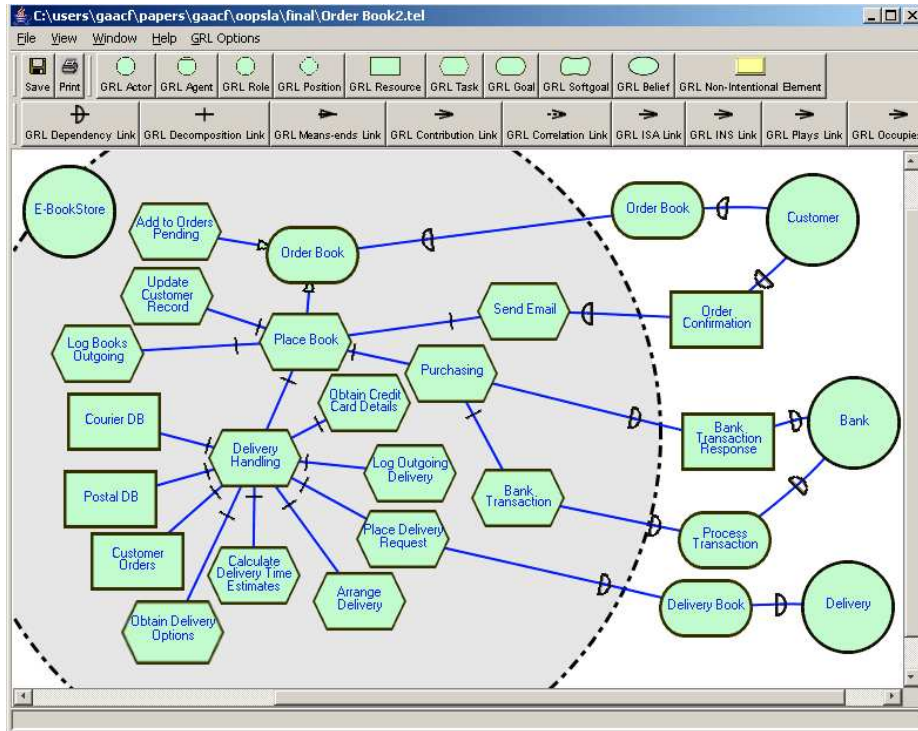


Fig. 1. Strategic Dependency Model

In order to illustrate consider part of  $i^*$  model for an electronic bookstore shown in Figure 1 (BookStore). This model has been developed based in the case study presented in [20]. The bookstore provides a web based selling channel from which customers can buy books on the web. In the initial stage, the strategic goals that represent the most general goal of the business are identified and the dependencies between the actors to achieve their goals are represented.



**Fig. 2.** Strategic Dependency Model

The other actors to the BookStore model are:

- *Customer* – person who buys books from BookStore;
- *E-BookStore* - multi-agent system that processes the Internet orders for the Bookstore
- *Stock Manager* - person who manages the stock of the BookStore;
- *Publisher* - company that provides books to the BookStore;
- *Card Processing Company* – external organisation that processes credit card transactions on behalf of BookStore;
- *Delivery* – external organisation that collects the books and then delivers the books to the Customer.
- *Bank* – company that processes the on-line transaction of the BookStore;
- *Telecom Cpy* - company that provides Internet services to the BookStore.

Figure 2 shows the partial SR model for the actor *E-Bookstore* representing a multi-agent system. The model shows how the multi-agent system will achieve the goal *Order Book* to actor *Customer*. The goal *Order Book* is realised through task *Place Book* that represents a successful order, or through task *Add to Orders Pending* that adds the order to an orders pending list when the book requested is not available. The task *Place Book* is decomposed in tasks *Delivery Handling*, that manages delivers orders to customers, *Update Customer Record* that updates the customer database, *Log Books Outgoing* that updates the stock database, *Purchasing* that manages the credit card transaction, and *Send Email* that informs the customer about the delivery details.

## 2.2 Prometheus

Prometheus methodology [20] has been developed over the past seven years and has been used in both industrial and academic settings. In Prometheus, goals are specified using a textual notation combined with a simple graph notation that shows the decomposition of the goal into sub-goals. The goals are identified from the initial description of the system and are refined in sub-goals using the technique of asking "how" it is possible to achieve the main goals.

Prometheus is an iterative methodology that is composed by three phases: *system specification*, *architectural design*, and *detailed design*. The system specification focuses on (a) identifying the system goals, the basic functionality of the system, the interface between the system and its environment in terms of inputs (percepts) and outputs (actions), and (b) developing use case scenarios that are similar to scenarios used in object-oriented approaches with a slightly enhanced structure. The architectural design phase focuses on (a) deciding what agent types the system will contain, (b) developing the agent descriptors, (c) capturing the structure of the system by using a system overview diagram that models relationships between agents, events, and shared data objects, and (d) describing the dynamic behaviour of the system. The behaviour of the system is described using interaction diagrams and interaction protocols. The detailed design is concerned with describing the agents in terms of capabilities, events, plans, and data structures. The artefacts generated in this phase are agent overview diagrams, capability overview diagrams, and descriptors for plan, data, and events. More details about Prometheus methodology can be found in [20].

Figure 3 presents an example of the use case scenario for BookStore taken from [20]. Examples of Delivery Handling and Purchasing functionality, Bank Transaction Response percept, Send Email and Place Delivery Request actions, and Courier DB, Postal DB, and Customer Orders data also taken from [20] is shown in Figure 4.

#	Type	Name	Functionality	Data
1	Goal	Obtain delivery options	Delivery handling	Uses: Courier DB, Postal DB
2	Goal	Calculate delivery time estimates	Delivery handling	Customer order record, Courier DB, Postal DB Produces: time estimates
3	Goal	Present information	Online interaction	Uses: temporary data
4	Percept	User input	Online interaction	Get delivery choice
5	Goal	Obtain credit card details	Purchasing	Uses: Customer DB
6	Percept	User input	Online interaction	Uses: Customer DB Produces: CC details (if needed), agreed transaction
7	Action	Bank transaction	Purchasing	Transaction details (temp)
8	Percept	Bank transaction response	Purchasing	
9	Goal	Arrange delivery	Delivery handling	Produces: Customer Orders, Uses and produces: customer order record
10	Action	Place delivery request	Delivery handling	Uses: customer order record
11	Goal	Log outgoing delivery	Delivery handling	Produces: Customer Orders
12	Goal	Log books outgoing	Stock management	uses: customer order record Produces: Stock DB
13	Goal	Update customer record	Profile monitor	Produces: Customer DB
14	Action	Send email	Customer contact	uses: Customer DB
<b>Variation 1:</b> Book is not currently available. Include information with delivery options. Replace steps 7-12 with steps to add the order to an orders pending file.				

**Fig. 3. Order Book Use Case Scenario**

<p><b>Functionality:</b> Delivery Handling  <b>Description:</b> This functionality manages delivery of orders to customers  <b>Triggers:</b>  <b>Actions:</b> Place delivery request  <b>Information used:</b> Courier DB, Postal DB, Customer Orders  <b>Information produced:</b> Customer orders  <b>Goals:</b> Obtain delivery options, Calculate delivery time estimates, Arrange delivery, Place delivery request, Log outgoing delivery, Determine delivery status, Fill pending order</p>	<p><b>Data:</b> Courier DB  <b>Description:</b> Contains information about courier companies, areas and rates</p>
<p><b>Percept:</b> Bank transaction response  <b>Description:</b> Response to request for credit card payment  <b>Information carried:</b> Accept/Reject, fraud (optional), amount, account ID  <b>Knowledge update:</b> none  <b>Source:</b> bank processing system  <b>Processing:</b> none  <b>Agents responding:</b> Sales Assistant  <b>Expect frequency:</b> Individual agent unlikely to receive more than 1 in total. Certainly no more than 1 every few minutes, maximum. System as whole could potentially receive 10 per minute maximum.</p>	<p><b>Functionality:</b> Purchasing  <b>Description:</b> Manage on-line sales of books, including credit card transaction  <b>Triggers:</b> Bank transaction response  <b>Actions:</b> Bank transaction  <b>Information used:</b> Customer Orders, Customer DB  <b>Information produced:</b> Purchase approval (temporary) (Customer DB, Customer Orders)  <b>Goals:</b> Obtain credit card details, Place order (online), Make payment (online)</p>
<p><b>Data:</b> Postal DB  <b>Description:</b> Contains information about postal rates.</p>	<p><b>Action:</b> Send Email  <b>Description:</b> Send email message (generic)  <b>Parameters:</b> address, content, sender address  <b>Duration:</b> Immediate  <b>Failure:</b> May bounce or fail to arrive.  <b>Partial change:</b> N/A  <b>Side effects:</b> None.</p>
<p><b>Data:</b> Customer Orders  <b>Description:</b> Contains information about customers, their profile, their history of visits to the site and orders, etc.</p>	<p><b>Action:</b> Place delivery request  <b>Description:</b> Send email request for delivery pick-up, either by courier, or by the postal room.  <b>Parameters:</b> email address (postal room, courier company, etc), delivery address, goods list  <b>Duration:</b> Immediate (action to send request is immediate – results will take time)  <b>Failure:</b> Mail may bounce. May also not be received without visible bounce.  <b>Partial change:</b> None  <b>Side effects:</b> None.</p>

**Fig. 4. Functionality, Actions, Data, and Percepts Descriptors**

### 3 Guidelines to Generate Prometheus from i\*

The guidelines defined in our approach are heuristics based on theoretic conceptual foundation of i\* technique and Prometheus methodology. These guidelines have been proposed based on deep analysis of the semantics of i\* and Prometheus system specification components. We present below these guidelines, followed by their explanation, suggestion of how to execute the guidelines (process), and examples based on the BookStore case study presented in Figures 1, 2, 3, and 4.

#### 3.1 Guidelines

**Guideline 1:** Goal dependencies in i\* SD model that are related to system actors are used to generate use case scenarios in Prometheus.

**Explanation:** Like in object-oriented approaches, in Prometheus use case scenarios represent the sequence of steps executed by the system to achieve main goals or functionality of the system. On the other hand, a goal dependency in i\* SD model is an objective that an actor related to it has to accomplish.

The existence of relationships between scenarios and organisational goals has also been suggested by other approaches [7, 12, 22, 23]. In [22], Ramesh and Jarke claim that scenarios are used to describe *Organizational Needs* and *System Objectives*. In [6], Cockburn presents a goal driven approach to create use case scenarios. He argues that "primary actor" initiates an interaction with the system to accomplish some goal.

**Process:** For each goal dependencies in which the system actor participates create a use case scenario.

**Example:** In our case study, the goal dependencies: *Query Late Books*, *Book Finding*, *Order Book*, *Order Status Query*, *Customer Profile Update*, *Provide Personalised Welcome*, *Authorize Payment*, *Process Transaction*, *New Catalogue*, *Cheap Price Notification*, *Stock Delayed*, *Stock Arrival*, *Pending Order Arrives*, *Delivery Book*, and *Missed Stock Arrival* will derive use case scenarios.

**Guideline 2:** Leaves tasks in i\* SR model are used to derive steps in Prometheus use case scenarios

**Explanation:** Primary goals identified in the SD model are refined in sub-components in the SR model through a top-down strategy that describes how the refined goal can be achieved. The leaves in the SR model representing tasks define how the system actor will execute the primary goals. On the other hand, the steps of use case scenarios represent activities executed by the system and how the system interacts with the actors (external to the system). In Prometheus the steps can be classified as goals, actions, and percepts. Actions and percepts represent how the system interacts with the environment (external actors), while goals are plans that an agent commit to execute in order to achieve a certain state of affair, keep some properties, or perform some activity.

**Process:** To identify the steps of the use case scenarios look at how the primary goal link dependencies achieved by the system actor are refined. The leaves tasks derived from the primary goals are mapped on steps of the use case scenario identified by applying Guideline 1. If a leaf task is linked with a link dependency in the SD model, then this leaf task should generate a step action in the use case scenario. Otherwise, a leaf task should generate a goal step in the use case scenario.

**Example:** In our case study for example the tasks: *Obtain Delivery Options*, *Calculate Delivery Time Estimates*, *Obtain Credit Card Details*, *Arrange Delivery*, *Log Outgoing Delivery*, *Log Books Outgoing*, and *Update Customer Record* are mapped to goal steps in use case scenario *Order Book*. The tasks: *Bank Transaction* and *Place Delivery Request* are mapped to action steps in use case scenario *Order Book* (see Figure 3).

**Guideline 3:** Resource dependencies in i\* SD model are used to generate percepts in Prometheus use case scenarios, when the system actor depends on other actors for the availability of these resources.

**Explanation:** Percepts are the agent's perceptual inputs at any given instant. In the SD model, resource dependency is a relationship in which an actor depends on another actor for the availability of an entity (physical or information).

**Process:** For each resource dependencies in which the system actor depends on another actor for the availability of the resource, create a percept. To identify the use case to which the percept is part of it is necessary to do a bottom-up analysis to find the primary goal that is related to the resource. For example, the resource *Order Confirmation* is mapped as percept in Prometheus. In order to identify which use case *Order Confirmation* percept is part of, we have to navigate through the network of relationships starting from task *Send Email* until reaching the primary goal *Order Book*.

**Example:** In our case study the resource *Bank Transaction Response* generates a percept that is part of the use case scenario *Order Book* (see Figure 3).

**Guideline 4:** Resources in the SR model can generate data in Prometheus.

**Explanation:** Data in Prometheus represent information used or produced by use case scenarios or functionality. Resources in the SR models represent necessary information to execute a task.

**Process:** For each resource in the system actor SR model, create a data in Prometheus.

**Example:** The resources *Courier DB*, *Postal DB*, and *Customer Order* in the SR model (see Figure 2) generate data in Prometheus (see Figure 4).

**Guideline 5:** Functionality in Prometheus can be derived from tasks in the SR model.

**Explanation:** Based on guidelines 2, 3, and 4 above and the facts that (a) functionality in Prometheus represent behaviour that includes a group of set related

goals, percepts, actions, and data relevant to the behaviour, and (b) a task in the SR model can be decomposed in sub-components of type resources, goals, and tasks.

**Process:** To identify functionality in Prometheus look for tasks that are decomposed in sub-elements in the system actor SR model.

**Example:** In our case study, the tasks *Purchasing* and *Delivery Handling* generate functionality in Prometheus (see Figure 4).

**Guideline 6:** Means-End links in the SR model generate variations of use case scenarios.

**Explanation:** Means-End represents alternatives (means) to achieve a goal (end). Variations are alternative scenarios to a use case.

**Process:** Choose one mean to be the principal scenario and for each additional mean in the SR model create a variation in the use case scenario.

**Example:** In our case study, the goal *Order Book* (end) can be achieved through the task *Place Order* that represents normal order, or through the task *Add to Order Pending* that represents an order when the book is not available (see Figure 3).

### 3.2 Discussion

The guidelines presented above should be considered as an initial attempt to support the enhancement of Prometheus with goals and business modelling (early requirements stage). However, it has to be appreciated that the *i\** technique and system specification phase of the Prometheus methodology represent different levels of abstraction in the software development life-cycle. Therefore, not every concept in *i\** can be used to generate elements in Prometheus system specification artefacts, and not every element in Prometheus system specification artefacts are derived from *i\** concepts. For example, steps 3 and 4 of use case scenario *Order Book* in Figure 3 are not represented in the *i\** model in Figure 2. In general, use case scenarios contain detailed information of how the system interacts with its users, while *i\** models represent how the system achieves a certain goal in a more abstract level.

Another issue is related to the fact that *i\** technique does not provide mechanisms to describe the temporal order of the intentional elements (although some extensions are proposed in [21]). This requires further analysis to identify the sequence of steps in use case scenarios.

## 4 Related Work

In recent years various approaches have been proposed to support goals [1, 3, 10] and business modelling [6, 13, 15, 29]. Some of these approaches have also been used to support object oriented software development [1, 15, 29].

The Rational Unified Process [15] provides an approach for business modelling that uses a business use case model and a business analysis model to define the processes, roles, and responsibilities of an organisation. RUP also provides guidelines

of how to generate use case and design models from business models in the context of object oriented software development. Although some agent-oriented methodologies such as ADELFE[2] and MaSE [11] are based on RUP, to the best of our knowledge, none of these methodologies incorporate the business-modelling phase in their approaches. Besides, ADELFE and MaSE provide weak support for some important concepts of intelligent agent such as belief, desire, and intention.

In the KAOS methodology [10], the business and strategic goals are represented as goal diagrams. The goal diagram is a set of interrelated diagrams linked through directed graphs where the roots represent business and strategic goals while the leaves represent the system requirements or *expectation*. The KAOS methodology provides the operation model in which the requirements identified in the goal model are used to identify the operations of the system. The operations can be discovered looking at the transitions of state that are generated by accomplishing the goals. To each transition an operation should be identified. The KAOS methodology could be customised to be applied in Prometheus methodology, but KAOS does not provide mechanisms to represent the structure of the business, and as a consequence, does not allow support for other types of analyses such as business process reengineering analysis, dependency analysis, and workflow analysis that are important for the understanding of the organisation by software developers.

In [17, 23], the authors introduced guidelines of how to translate the business model represented in *i\** into a compatible UML use case specification. However, these approaches can not be used with Prometheus use case scenarios. The use case scenarios in Prometheus are more detailed than use case scenarios used by the approaches above and they are not able to represent concepts used in Prometheus like actions, percepts, data, and functionality.

Various methodologies have been developed to support agent oriented development [4, 5, 11, 14, 28]. In some agent oriented methodologies like Tropos [4, 14] and MESSAGE [5] the authors have recognised that business modelling is an important phase during software system development and have proposed mechanisms to describe the organisational context. However, none of the existing methodologies provide detailed support to all phases of the software development process.

## 5 Conclusions and Future Work

This paper presents an approach to refine Prometheus methodology with goals and business modelling, and proposes some guidelines of how to generate Prometheus system specification artefacts from organisational models expressed in *i\**. The approach helps developers to identify in a systematic way the requirements of the system to support the functions of the organisation.

Currently we are validating our work by the use of other case studies and developing a prototype tool to support automatic generation of system specification artefacts based on the guidelines being proposed. We are also extending our work to support generation of artefacts created in the architectural and detailed design phases of Prometheus methodology. We plan to propose new guidelines to support the

generation of these artefacts based on goals and business models and incorporate these guidelines in the prototype tool being developed.

The work presented in this paper is part of a large programme of research to allow automatic generation of traceability relations between artefacts created during object and agent oriented software development. Initial work to allow traceability generation in agent oriented systems has been proposed in [8]. This work is based on our previous work to allow automatic traceability generation in object oriented requirements specifications [24][30]. The approach proposed in [8][24][30] is a rule-based approach in which traceability relations are automatically generated based on pre-defined rules. We intend to specify the guidelines presented in this paper and the new guidelines under development to the architectural and detailed design phases as rules, and use our rule-based approach to generate traceability relations between the different artefacts. Our work is not limited to the early phases of the software development life-cycle. We also plan to propose rules to support generation of traceability relations between artefacts generated in later phases of the development life-cycle (e.g. source code, test cases).

## References

1. Anton, A.: Goal Based Requirements Analysis. In Proceeding Second International Conference on Requirement Engineering - ICRE (1996)
2. Bernon, C., Gleizes, M., Peyruqueou, S., Picard, G.: ADELFE, a Methodology for Adaptive Multi-Agent Systems Engineering. Third International Workshop on Engineering Societies in the Agents World (ESAW-2002), Madrid, (2002).
3. Bochini, D., Paolini, P.: Capturing Web Application Requirement through Goal-Oriented Analysis. In Proceedings of the Workshop on Requirements Engineering - WER, Spain (2002)
4. Bresciani, P., Giorgini, P., Giunchiglia, F., and Mylopoulos, J., and Perini, A.: TROPOS: An Agent-Oriented Software Development Methodology. In Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers Volume 8, Issue 3, May (2004) 203 - 236
5. Caire, G., et al.: Agent Oriented Analysis Using MESSAGE/UML: Agent-Oriented Software Engineering II, M. Wooldridge, G. Wei, and P. Ciancarine, (eds.), Volume 2222 of LNCS, New York: Springer (2001) 119-135
6. Cesare, S., Lycett, M. and Patel, D.: Business Modelling with UML: Distilling Directions for Future Research. ICEIS (2002) 570-579.
7. Cockburn A.: Structuring Use Case with Goal, Writing Effective Use Cases, Addison-Wesley (2001).
8. Cysneiros G., Zisman A., and Spanoudakis G: A Traceability Approach for i\* and UML Models. Proceedings of 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems - ICSE 2003 , Oregon, USA, May (2003).
9. Dam, K., Winikoff, M.: Comparing Agent-Oriented Methodologies. Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems, Melbourne, July (AAMAS03).
10. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal Directed Requirements Acquisition, Science of Computer Programming. Vol. 20 (1993) 3-50.

11. DeLoach, S.: Analysis and design using MaSE and agent Tool. 12<sup>th</sup> Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001), Ohio, (2001)
12. Estrada, H., Martínez A., Pastor, O.: Goal-based business modeling oriented towards late requirements generation. In Proceeding of the International Conference on Conceptual Modeling - ER (2003) 277-290
13. Eriksson, H. and Penker, M.: Business Modeling with UML: business patterns at work, John Wiley & Sons (2000)
14. Giunchiglia, F., Mylopoulos, J., and Perini, A: The TROPOS Software Development Methodology: Processes, Models and Diagrams. Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems AAMAS'02, C. Castelfranchi and W. Johnsons, (eds.), New York: ACM Press (2002) 35-36
15. Kruchten, P.: Rational Unified Process: An Introduction. Addison-Wesley (2003)
16. Luck, M., McBurney, P. and Preist, C.: Agent Technology: Enabling Next Generation Computing. AgentLink (2003)
17. Martínez, A., Pastor, O., and Estrada, H.: Closing the gap between Organisational Modeling and Information System Modeling. WER (2003) 93-108
18. Odell, J., Parunak, V., and Bauer, B.: Representing Agent Interaction Protocols in UML. In: *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge Eds., Springer-Verlag, Berlin, (2001) 121–140
19. OMG: UML 2.0 Superstructure Specification, OMG document ptc/03-08-02, September 2, (2003)
20. Padgham, L. and Winikoff, M.: Developing Intelligent Agent Systems - A practical guide, John Wiley & Sons (2004)
21. Perini, A., Pistore, M., Roveri, M. and Susi, A.: Agent-oriented modeling by interleaving formal and informal specification. Agent Oriented Software Engineering (AOSE-2003), Melbourne, Australia (2003).
22. Ramesh, B. and Mathias J.: Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering Journal, Vol. 27, No.1, January (2001)
23. Santander, V., Castro J.: Deriving Use Cases from Organizational Modeling. RE02 - IEEE Joint Conference on Requirements Engineering, Essen, Germany (2002)
24. Spanoudakis G., Zisman A., Perez-Minana E., Krause P., "Rule-Based Generation of Requirements Traceability Relations", Journal of Systems and Software, vol. 5, (2004)
25. Sudeikat, J., Braubach, L., Pokahr, A., and Lamersdorf, W. Evaluation of Agent-Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform. Workshop on Agent-Oriented Software Engineering (AOSE-2004)
26. Weiß, G.: Agent orientation in software engineering. Knowledge Engineering Review. Vol. 16(4), (2002) 349-373
27. Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley & Sons Ltd, England (2002)
28. Wooldridge, M., Jennings, N., and Kinny, D.: The Gaia Methodology for Agent-Oriented and Design. Autonomous Agents and Multi-Agent Systems, Vol. 3, No. 3, (2000) 285-312
29. Yu, E.: Modelling Strategic Relationships for Process Reengineering. PhD thesis, University of Toronto, Department of Computer Science (1995)
30. Zisman A., Spanoudakis G., Perez-Minana E., Krause P.: Tracing Software Requirements Artefacts", The 2003 International Conference on Software Engineering Research and Practice (SERP'03), Las Vegas, Nevada, USA, June (2003).