University of Groningen

Applying the Tropos Methodology for Analysing Web Services Requirements and Reasoning about Qualities of Services

Aiello, Marco; Giorgini, Paolo

Published in:
Upgrade: The European Journal for the Informatics Professional

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2004

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
Aiello, M., & Giorgini, P. (2004). Applying the Tropos Methodology for Analysing Web Services Requirements and Reasoning about Qualities of Services. *Upgrade: The European Journal for the Informatics Professional*, *5*(4), 20-26.

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license. More information can be found on the University of Groningen website: https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment.

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Applying the Tropos Methodology for Analysing Web Services Requirements and Reasoning about Qualities of Services

Marco Aiello and Paolo Giorgini

The shift in software engineering from the design, implementation and management of isolated software elements towards a network of autonomous interoperable service is calling for a shift in the way software is designed. We propose the use of the agent-oriented methodology Tropos for the analysis of Web service requirements. We show how the Tropos methodology adapts to the case of Web services and in particular how it can be used to model quality of service requirements. We base the investigation on a representative case study in the retail industry.

Keywords: Agent-oriented Methodology, Software Engineering, Quality of Service, Web Services.

1 Introduction

The opportunities offered by the growth of the Internet in terms of networking infrastructure, open standards and reach of users, are focusing research and industrial interests on application areas such as electronic commerce, enterprise resource planning, supply-chain management, and peer-to-peer computing, to name the most prominent ones. This is deeply and irreversibly changing our views of software and, in particular, software engineering. Interoperability and scalability play a fundamental role in the development and management of software as nowadays a piece of software cannot be thought in isolation, but rather, as an element of a network of interacting software elements. Continuous evolution to meet changing and new requirements is becoming an essential feature of software. In addition, software must be robust and autonomous, capable of serving end users with a minimum of overheads and interference. Software is thus becoming more and more a service offered to a human user or to another software element rather than an isolated application running on a specific machine for a specific predefined requirement. This is the view of software as a 'service' well conceptualized by the service-oriented computing paradigm [16].

The most prominent example of the service-oriented computing paradigm is to be found on the Internet, where the set of standard interfaces for the interaction of software elements is well-known as Web services. Web services are a set of standardized interfaces for the description, discovery, invocation, composition and orchestration of independent loosely-coupled software elements residing on the Internet. As software is changing, one of the challenges is to find appropriate concepts, tools and techniques to design, engineer and manage software. Traditional software engineering methods may prove to be cumbersome or not capable of capturing the full potential of the service-oriented paradigm. In [6], UML is used to design business processes that manage the execution and interaction with various independent Web services. Aspect-oriented programming is investigated in [10] for designing Web service based electronic utilities, i.e., distributed applications. But all these approaches lack fundamental features of Web services, that is, the autonomy of services, the need to model services at a high level of abstraction in terms of what a Web services goal is rather than all its atomic functionalities, and the need for run-time support for changing execution environments.

Agent-oriented software development methodologies are gaining popularity over traditional software development

Marco Aiello is an Assistant Professor at the Department of Information and Communication Technologies (DIT) at University of Trento, Italy. Head of the Distributed Systems and Service Oriented Computing research program at DIT, his current research interests are in the area of service-oriented computing and distributed systems. In 2002 he has obtained a PhD from the University of Amsterdam (Nederlands) defending a thesis on spatial reasoning. He holds a laurea degree in Ingegneria Informatica from University of Rome La Sapienza, Italy. He has published over 20 articles in international journals, books, magazine and conferences. In 2003 he has won the best dissertation award from the Italian Association for Artificial Intelligence (AI*IA). Since 1999 he is the information director of the ACM Transactions on Computational Logic. <aiellom@dit.unitn.it>

Paolo Giorgini is an Assistant Professor in Computer Science at University of Trento, Italy. He received his Ph.D. degree from Computer Science Institute of University of Ancona, Italy, in 1998. His research interests include agent-oriented software engineering, multi-agent system modelling and design, knowledge representation and conceptual modelling. He has worked on the development of requirements and design languages for agent-based systems, and the application of knowledge representation techniques to software repositories and software development. His publication list includes refereed journal and conference proceedings papers and four edited books. He has contributed to the organization of international conferences as chair and program committee member, such as CoopIS, ER, CAiSE, AAMAS, EUMAS, AOSE, AOIS, and ISWC. <paolo.giorgini@dit.unitn.it>

 © Novática

approaches [11][7]. After all, agent-based architectures do provide for an open, evolving architecture that can change at runtime to exploit the services of new agents, or replace underperforming ones. In addition, software agents can, in principle, cope with unforeseen circumstances because their architecture includes goals along with a planning capability for meeting them.

Mostly, software is thought of in terms of its functional behaviour, i.e., what the software does. But this does not completely describe the software's behaviour. Non-functional properties, such as the average execution time, are important elements to determine the usability and utility of a software product. With the term Quality of Service (QoS, for short), we refer to the non-functional properties of a software service. In the context of Web services, QoS is a critical task for a number of reasons: first, autonomous services depend on one another for their functioning and they need to be aware of the QoS of the collaborating services; second, services may compete with one another and a service requester may decide on a service based on its QoS properties; third, a service provider may offer the same function with differentiated QoS, for instance at different prices, and must therefore publicize the different qualities of the same function. There is no consensus on what the qualities are that fall in the QoS of a Web service. The traditional view inherited from the networking community places only performance and availability in the set of QoS, but other properties are also relevant such as accessibility, integrity, reliability, regulatory and security [14]. Some authors assume that any custom parameter that can be modelled as a nonfunctional property of a service may be considered as an element of the QoS [18, 19, 15]. In this paper, we consider a wide spectrum of QoS properties, such as performance, cost, reliability and security, and we introduce a framework that is flexible and open to any user-defined quality as QoS.

We propose the use of the agent-oriented methodology Tropos [3][1] for the analysis of Web service requirements. The goal is to model software deployed using Web services at a high abstraction level by using agent-oriented techniques. In this approach, we do not model every individual Web service as one agent, but rather model the whole set of interacting services as a multi-agent system, where different dependent strong and soft goals coexist. Thus, using Tropos we consider Web services not simply on the basis of their functional interfaces, but rather on how each service impacts shared goals of a software system, and which role each service has in such an entire system. These considerations are particularly useful to model complex systems through their life-cycle, that is, from the early requirements analysis down to the actual implementation. In particular, the methodology emphasizes early requirements analysis, the phase that precedes the prescriptive requirements

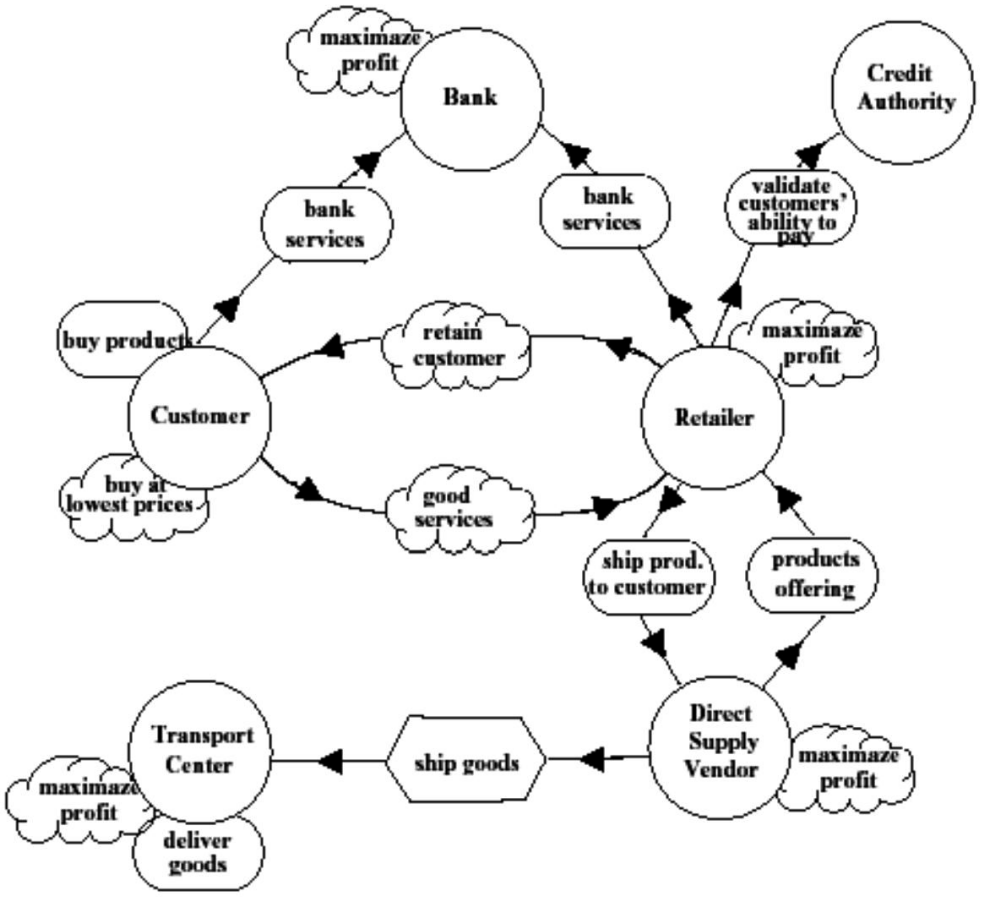

Figure 1: Actor Diagram for the Online Retail Store Example.

© Novática

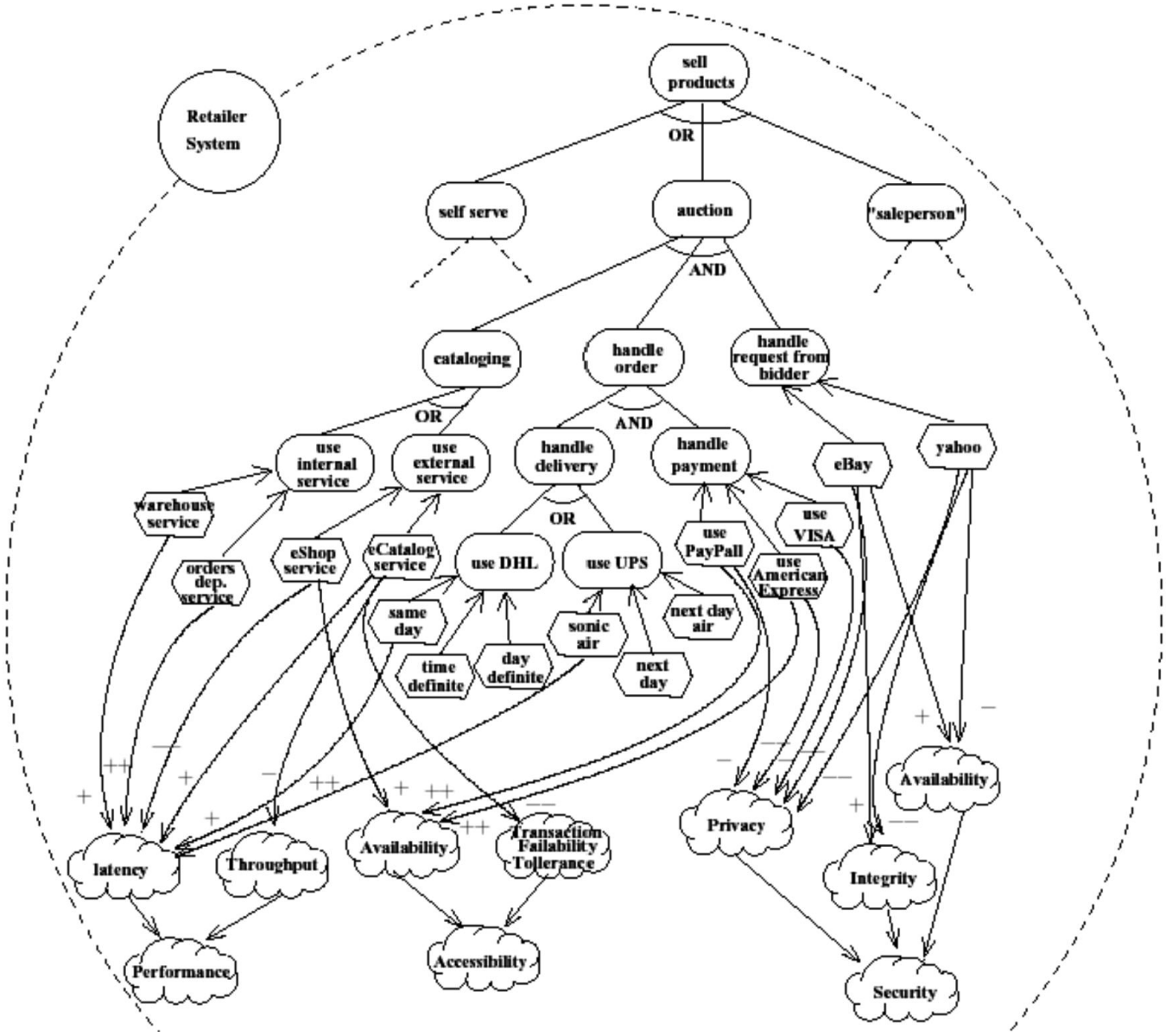

Figure 2: Part of the Rationale Diagram for the Retailer System.

specification. In this paper we use this feature of Tropos to model quality of services of systems based on Web services as QoS are typically high level characteristics not described at the level of Web service individual interfaces.

The paper is structured as follows: Section 2 introduces the Tropos methodology, where concepts, modelling and analysis techniques are presented through a representative case study. In Section 3 we propose a goal analysis framework for qualitative and quantitative reasoning about QoS. Section 4 discusses how the framework relates to existing Web service standards. Concluding remarks are summarized in Section 5.

2 Requirements Analysis with Tropos

Tropos rests on the idea of using requirements modelling concepts to build a model of the system-to-be within its operational environment. This model is incrementally refined and extended, providing a common interface to the various software development activities. The model also serves as a basis for documentation and evolution of the software system.

Tropos is an agent-oriented methodology in which agent's related notions (such as agent, goal, belief task, and social dependency) [4] are used in the whole development software process, from early phases of the requirements analysis. Requirements analysis in Tropos consists of two phases: *Early Requirements* and *Late Requirements* analysis. Early requirements is concerned with understanding the organizational context within which the system-to-be will eventually function. Late requirements analysis, on the other hand, is concerned with a definition of the functional and non-functional requirements of the system-to-be.

© Novática

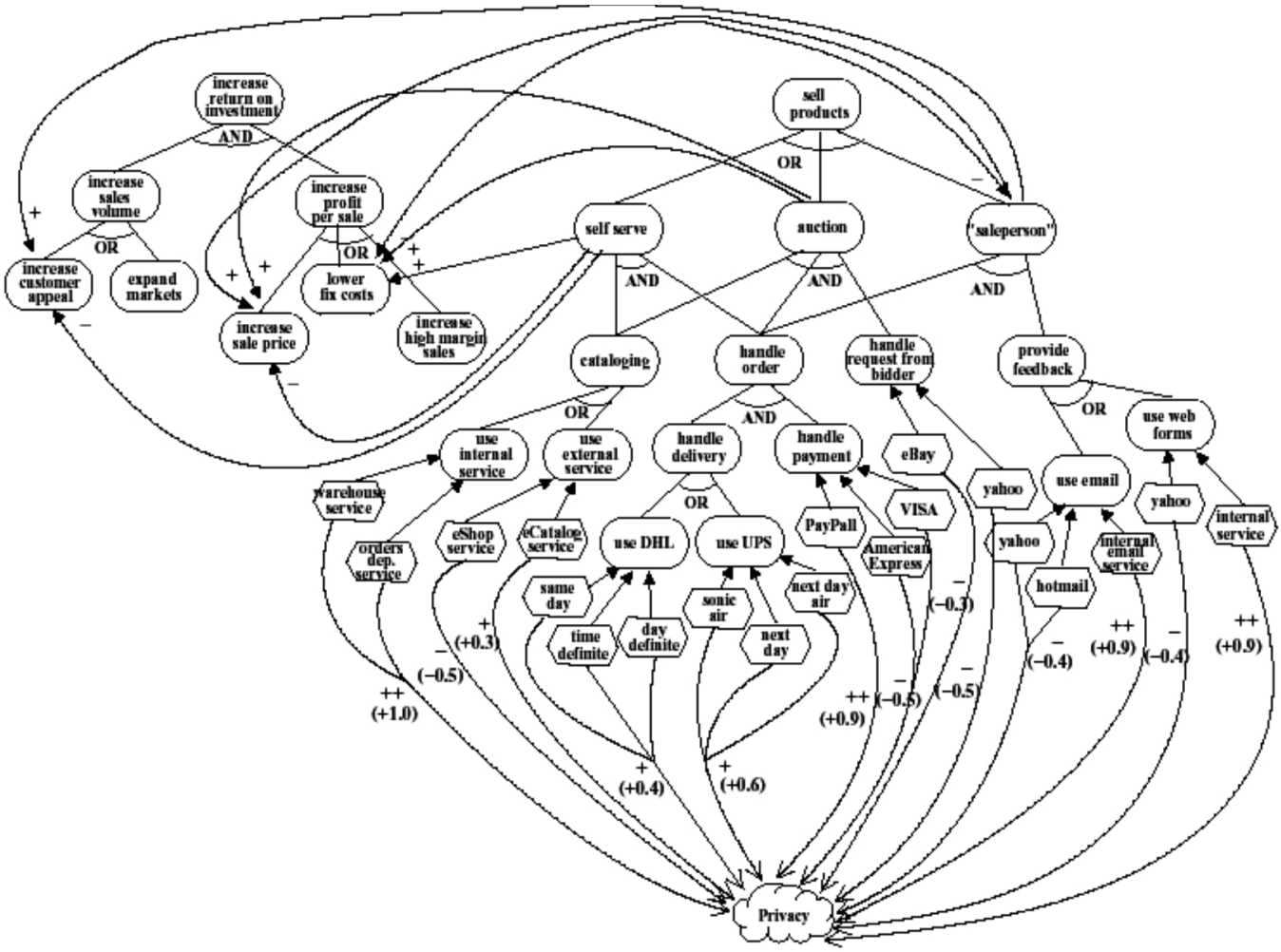

Figure 3: Partial Goal Model Used for the Reasoning about Qualities.

During early requirements analysis, the requirements engineer identifies the domain stakeholders and models them as social actors, who depend on one another for goals to be fulfilled, tasks to be performed, and resources to be furnished. Through these dependencies, one can answer *why* questions, besides *what* and *how*, regarding system functionality. Answers to why questions ultimately link system functionality to stakeholder needs, preferences and objectives. Actor diagrams and rationale diagrams are used in this phase.

Figure 1 shows the actor diagram for an online retail store example. This example is an extended and revised version of the example introduced in [12]. The diagrams present the principal stakeholders and their interests. The *Customer* actor has the goal to *buy products* and the softgoal to *buy at lowest prices*. It depends on the *Retailer* actor for having *good services* and on the *Bank* for using *bank services*. The *Retailer* actor has the softgoal[1] of *maximizing profit* and depends on the *Bank* for the *bank services*, on the *CreditAuthority* to *validate the customers' ability to pay*, and on the *Direct Supply Vendor* to *ship products to the customers*. The *Direct Supply Vendor* depends on the *Retailer* for *products offering* and on the *Transport Centre* to *ship goods*.

Actor diagrams are extended during early requirements analysis by incrementally adding more specific actor dependencies which come out from a means-end analysis of each goal. This analysis is specified using rationale diagrams, which appear as a balloon within which goals of a specific actor are analysed and dependencies with other actors are established. Goals are decomposed into subgoals and positive/negative contributions of subgoals to goals are specified (see Figure 2 as an example.)

During *late requirements* analysis, the conceptual model developed during early requirements is extended to include the system-to-be as a new actor, along with dependencies between this actor and others in its environment. These dependencies define functional and non-functional requirements for the system-to-be. Actor diagrams and rationale diagrams are used also in this phase.

Figure 2 shows part of the rationale diagram for the retailer system. Basically, this analysis extends the goal analysis for the

1. Softgoals represent vaguely defined goals, with no clear-cut criteria for their fulfilment

© Novática

Retailer actor. The extension includes the analysis of the services (hexagons) that can be adopted in order to satisfy the *Retailer System's* goals and how such services impact on the qualities of the system, namely the softgoals *security*, *reliability*, and *performances*. For the sake of simplicity, the diagram reports only some of the possible contribution links between services and qualities.

3 Reasoning about Qualities

The adoption of specific Web services can have different consequences on the qualities of the software system. In order to reason about these effects, we propose to adapt and use the Tropos goal analysis techniques presented in [8][17].

The analysis starts from the goal models developed in the late requirements phase, and in particular focuses on the models developed for the system we want to develop (the Retailer System in our example). The goal model consists of a set of nodes (goals and services/tasks) and relations over them, including the n-ary relations AND, OR and the binary relations (contribution links) + and –. + (–) relationship is used to represent that the satisfaction of a goal contributes to the satisfaction (denial) of another goal.

Figure 3 presents part of the goal model for the Retailer actor extended with the analysis concerning the adoption of the adopted services and their impact on the qualities of the system. The contribution links between services and privacy quality are both qualitative and quantitative. Qualitative links are used for qualitative reasoning, while quantitative links for quantitative reasoning.

3.1 Forward Reasoning.

An algorithm and its implementation for forward reasoning has been presented in [8]. Given a goal graph, the user assigns some initial values to some goals (typically leaf goals), then these values are forward propagated to all other goals according to the rules above described in [8]. The initial values can be qualitative (such as fully satisfied, partially satisfied, fully denied, and partially denied) or quantitative (probability for the satisfaction or the denial of a goal).

The user can then check the final values of the goals of interest (typically root goals), and see whether she/he is satisfied with these values. Basically, forward reasoning is used to observe the effects of the adoption of a set of Web services.

For the example in Figure 3, we might, for instance, be interested in finding the effects of adopting a set of services over the top goals *increase return on investment* and *sell products* as well as over the privacy softgoal *Security*. So for instance, suppose we decide to use the following services (i.e., we assign FS (Fully Satisfied) labels to them): *eShop service*, *day definite* (by DHL), and *PayPall*. With these services *Privacy* will be partially satisfied (PS), *sell products* will be fully satisfied (FS), and *increase return on investment* will be partially denied (PD). Of course, we could suppose that *expand the markets* and then *increase return on investment* will also be partially satisfied. In this case we have a conflict, that is, we have evidence both for the satisfaction and the denial of the goal.

Similarly, we can perform qualitative reasoning. Suppose to assign to *eShop service*, *day definite* (by DHL), and *PayPall* the value S=1 (i.e., probability of Satisfaction equal to 1), that is, we decide to adopt them. As consequence we obtain that *Privacy* will assume S= 0.4 and D=0.5 (i.e., probability of denial equal to 0.5) for denial, whereas *sell products* will assume S=1.0. This means we can satisfy our top goal but we cannot say much about *Privacy* (we have a conflict). Note that we have not considered the effects produced by the adopted services on the *increase return on investment* goal.

3.2 Backward Reasoning

An algorithm and its implementation of backward reasoning has been presented in [17,5]. In particular, the implemented tool solves the following two problems: (1) find an initial assignment of labels to leaf goals which satisfies a desired final status of root goals by upward value propagation, while respecting some given constraints; and (2) find a minimum cost assignment of labels to leaf goals which satisfies root goals. Basically, backward reasoning is used to find a possible set of services to be adopted in order to satisfy our top goals at the minimum cost.

In our example, we might be interested in finding a set of services (at the minimum cost) that satisfy our top goal *sell products* and *Privacy* softgoals. For instance, suppose we want to satisfy *sell products* and at least partially satisfy *Privacy* softgoal. The software gives no solution for the full satisfaction of softgoal *Privacy*, but it produces several solutions for its partial satisfaction. Fixing the same cost for all services, one of these solutions consists of the following services: *same day* (by DHL), *PayPall*, and *internal email service*.

Analogously for the quantitative reasoning, we might be interested in finding the minimal cost set of services that satisfy totally the top goal *sell products* (S=1.0) and partially the *Privacy* softgoals, Say S=0.7. The software produces a set of solutions that satisfy such a request, i.e., a set of services that, if adopted (S=1.0), produce the desired results over *sell products* and *Privacy*.

4 Quality-aware Web Services

The qualities of the retailer system we have presented so far are performance, reliability and security. These are custom QoS measures for software and, in particular, for services, but are not the only possible ones. Availability, integrity, and regulatory conformance are other QoS that may need to be modelled, furthermore, other parameters specific to the application at hand may need modelling. The framework proposed does not commit to any specific quality, but rather gives freedom of choice to the designer.

This freedom must be reflected at the service level, in other words, services must be able to 2describe their qualities and have shared vocabulary of service qualities. Standard service description languages such as WSDL (Web Services Description Language) lack the necessary constructs to address this issue. Two approaches are possible: on the one hand, one can extend WSDL with ports for the description of quality properties of the services (such as in [9]); on the other hand, one could

© Novática

complement WSDL interfaces with ancillary documentation for the description of quality of service characteristics of the service.

In [15] a symmetric model based on constraint satisfaction techniques is used to verify QoS desires coming from the requester. In [19], an XML-based language (eXtensible Markup Language) used for negotiating QoS values among service requester and provider is presented. A semantic web approach in which services are searched based on quality of service attributes semantically tagged is presented in [18]. A predictive QoS model for workflows involving QoS properties is proposed in [2]. In addition, the industry has proposed a number of standards to this end: IBM Web Service Level Agreement (WSLA) and HP's Web Service Management Language (WSML) are examples of languages used to describe quality metrics of services, [13]. What is missing is a framework to design composition and orchestrations of services with desired global QoS requirements and, dually, to analyse global QoS properties of Web service compositions.

The framework we propose is independent from the choice made on whether one extends WSDL or one uses an extra document for the description of service qualities such as WS-Policy. The only requirement is, naturally, that all the services implement the same infrastructure for service quality description and use a negotiated and agreed ontology for describing the qualities. Let us consider the Privacy quality of the Retailer System example by adapting WS-Policy to our framework. Suppose the PayPall service publishes the following policy:

```
01 <wsp:Policy xmlns:wsse="..."
   xmlns:wsp="...">
02 <wsp:ExactlyOne>
03    <wsse:SecurityToken wsp:Usage="wsp:
      Required" wsp:Preference="80">
04       <wsse:TokenType>wsse:SecureSocket
         LayerVersion3.1</wsse:TokenType>
05    </wsse:SecurityToken>
06    <wsse:SecurityToken wsp:Usage="wsp:
      Required" wsp:Preference="60">
07       <wsse:TokenType>wsse:SecureSocket
         LayerVersion3.0</wsse:TokenType>
08    </wsse:SecurityToken>
09    <wsse:SecurityToken wsp:Usage="wsp:
      Required" wsp:Preference="10">
10       <wsse:TokenType>wsse:SecureSocket
         LayerVersion2.0</wsse:TokenType>
11    </wsse:SecurityToken>
12 </wsp:ExactlyOne>
13 </wsp:Policy>
```

In our framework, this policy would be interpreted as the fact that the *PayPall* is providing its services with three different qualities, the three SecurityTokens. At least one of these needs to be chosen, line 02. The different quality of service of these is represented by the wsp: Preference attribute on lines 03, 06, 09. This is interpreted as the fact that the first choice (line 04) gives a contribution to *Privacy* of 0.8, while the second (line 07) of 0.6 and the third (line 10) of 0.1.

5 Concluding Remarks

The shift in software engineering from the design, implementation and management of isolated software elements towards a network of autonomous interoperable service is motivating the investigation of new modelling and design techniques. We have proposed the use of the agent-oriented methodology Tropos for the analysis of Web service requirements. We have shown how the Tropos methodology adapts to the case of Web services and in particular how it can be used to model quality of service requirements. Forward reasoning and goal satisfaction have been proposed to design an architecture meeting given QoS requirements, but also to understand which QoS properties a system will have and how the various services influence the QoS parameters. We have based our investigation on a representative case study and have shown a wide range of non-functional properties to be captured by the framework we propose. Finally, we have shown how one can adapt existing Web service technologies to be included in the proposed framework.

A tool exists for the analysis of early requirements using Tropos. The extension of such a tool to deal with Web services and in particular with QoS requirements for Web services is under way.

A limitation of the proposed approach is that the contribution of the single service to a given element of the quality of services et is independent from those of the other services, that is, the combined effect of different services to the same quality is not captured. Future investigation will be devoted to solving this issue by considering global interaction of quality features.

References

[1] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agents and Multi-Agent Systems, 8(3):203–236, May 2004.

[2] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and Web service processes. Journal of Web Semantics, 2004. To appear.

[3] J. Castro, M. Kolp, and J. Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems. Elsevier, Amsterdam, the Netherlands, (to appear).

[4] P. Cohen and H. Levesque. Intention is choice with commitment. Artificial Intelligence, 32(3):213–261, 1990.

[5] S. Fante. Goal-Oriented Requirements Engineering: tecniche Numeriche di Analisi Top-down and Bottom-up. Master's thesis, Department of Information and communication Technology – University of Trento, 2004.

[6] T. Gardner. UML modelling of automated business processes with a mapping to BPEL4WS. In G. Piccinelli and S. Weerawarana, editors, European workshop on OO and Web Service, 2003. IBM Research Report. IBM. Computer Science, (RA 220).

[7] P. Giorgini, J. Müller, and J. Odell, editors. Agent-Oriented Software Engineering. LNCS 2935. Springer, 2003.

© Novática

[8] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Formal reasoning techniques for goal models. Journal on Data Semantics, Springer, 2004.

[9] D. Gouscos, M. Kalikakis, and P. Georgiadis. An approach to modelling Web service qos and provision price. In 1st Web Services Quality Workshop (WQW2003) at WISE, 2003.

[10] B. Hailpern and P. Tarr. Software engineering for Web services: A focus on separation of concerns. Technical report, IBM Research Reports, 2001. RC22184 (W0109-054).

[11] N. R. Jennings. On agent-based software engineering. Artificial Intelligence, 117(2), 2000.

[12] D. Lau and J. Mylopoulos. Designing Web Services with Tropos. In Proceedings of the 2004 IEEE International Conference on Web Services, San Diego, California, USA, July 6–9 2004.

[13] H. Ludwig. Web services qos: External slas and internal policies or: How do we deliver what we promise? In 1st Web Services Quality Workshop (WQW2003) at WISE, 2003.

[14] A. Mani and A. Nagarajan. Understanding quality of service for Web services, 2002. http: //www-106.ibm.com/developerworks/library/ws-quality.html.

[15] O. Martn-Daz, A. R. Corts, A. Durn, D. Benavides, and M. Toro. Automating the procurement of Web services. In Service-Oriented Computing (ICSOC), pages 91–103. LNCS 2910, Springer, 2003.

[16] M. P. Papazoglou and D. Georgakopoulos. Service oriented computing. Communications of the ACM, 46(10), 2003.

[17] R. Sebastiani, P. Giorgini, and J. Mylopoulos. Simple and minimum-cost satisfiability for goal models. In Proceedings of the 16th Conference On Advanced Information Systems Engineering (CAiSE*04). LNCS, Springer, 2004.

[18] M. P. Singh and A. S. Bilgin. A DAML-based repository for qos-aware semantic Web service selection. In IEEE International Conference on Web Services (ICWS 2004), 2004.

[19] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. Schiller. A concept for qos integration in Web services. In 1st Web Services Quality Workshop (WQW2003) at WISE, 2003.

 © Novática