UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

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

Marco Aiello and Paolo Giorgini

May 2004

Technical Report # DIT-04-034

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

Marco Aiello and Paolo Giorgini
Department of Information and Communication Technology
University of Trento - Italy
{aiellom,paolo.giorgini}@dit.unitn.it

May 7, 2004

Abstract

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 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. We base the investigation on a representative case study in the retailing industry.

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 on 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. Software must also operate on different platforms, without recompilation, and with minimal assumptions about its operating environment and its users. As well, software must be robust and autonomous, capable of serving end users with a minimum of overhead 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 [18]. 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, 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 technique to design, engineer and manage software. Traditional software engineering methods may prove to be cumbersome or to not capture the full potential of the service-oriented paradigm. Differences between objects and services are, for instance, presented in [1]. In [8], UML is used to design business processes that manage the execution and interaction with various independent web services. Aspect-oriented programming is investigated in [12] for designing web service based electronic utilizes, 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 approaches [13, 9]. After all, agent-based architectures *do* provide for an open, evolving architecture that can change at run-time to exploit the services of new agents, or replace under-performing 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.

Most often software is thought in terms of its functional behavior, i.e., what the software *does*. But this does not completely describe the software's behavior. Non-functional properties, such as for instance 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 one another for their functioning and they need to be aware of the QoS of the collaborating services; second, services may compete one another and a service requester may decide for 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 difference qualities of the same function.

There is no consensus on what are the qualities 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, security [16]. Some authors assume that any custom parameter that can be modeled as non-functional property of a service may be considered as an element of the QoS [20, 21, 17]. In this paper, we consider a wide spectrum of QoS properties, such as

performance, cost, reliability, 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 [4, 2] for the analysis of web service requirements. We show how this methodology is particularly well suited to reason about quality of service requirements for web services. We present a representative case study and show that a wide range of non-functional properties can be captured by the proposed framework.

Tropos is based on two key features. First, the notion of agent and related mentalistic notions are used in all software development phases, from the early requirements analysis down to the actual implementation. Second, the methodology emphasizes early requirements analysis, the phase that precedes the prescriptive requirements specification. In this respect, Tropos is quite different from other agent- and object-oriented software development methodologies.

Paying attention to the activities that precede the specification of prescriptive requirements for the system-to-be [6, 24] means that developers can capture and analyze the goals of stakeholders. These goals play a crucial role in defining the requirements for the new system. Put another way, prescriptive requirements capture the *what* and the *how* for the system-to-be. Early requirements, on the other hand, capture the reasons *why* a software system is developed. This new perspective, in turn, supports a more refined analysis of system dependencies and a more uniform treatment of functional and non-functional requirements.

The paper is structured as follows. Section 2 introduces the Tropos methodology, where concepts, modeling and analysis techniques are presented trough 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 modeling 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.

Requirement analysis represents the initial phase in most software engineering methodologies. 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.

Tropos adopts the i^* [24] modeling framework for analyzing requirements. In i^* (which stands for "distributed intentionality"), stakeholders are represented

as (social) actors who depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. The *i** framework includes the *strategic dependency model* (actor diagram in Tropos) for describing the network of inter-dependencies among actors, as well as the *strategic rationale model* (rationale diagram in Tropos) for describing and supporting the reasoning that each actor goes through concerning its relationships with other actors. These models have been formalized using intentional concepts from Artificial Intelligence, such as goal, belief, ability, and commitment (e.g., [5]). The framework has been presented in detail in [24] and has been related to different application areas, including requirements engineering [22], software processes [23], and business process reengineering [25].

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.

An actor diagram is a graph involving *actors* who have *strategic dependencies* among each other. A dependency represents an "agreement" (called *dependum*) between two actors: the *depender* and the *dependee*. The *depender* depends on the *dependee,* to deliver on the dependum. The dependum can be a *goal* to be fulfilled, a *task* to be performed, or a *resource* to be delivered. In addition, the depender may depend on the dependee for a *softgoal* to be fulfilled. Softgoals represent vaguely defined goals, with no clear-cut criteria for their fulfillment. Graphically, actors are represented as circles; dependums – goals, softgoals, tasks and resources – are respectively represented as ovals, clouds, hexagons and rectangles; and dependencies have the form *depender* → *dependum* → *dependee*.

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 [14]. The diagrams presents 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 use bank services. The Retailer actor has the softgoal of maximizing profit and depends on the Bank for the bank services (some of these services can be totally different from those of used by the customer), on the Credit Authority 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 Center to ship goods.

Actor diagrams are extended during early requirements analysis by incrementally adding more specific actor dependencies which come out from a means-ends analysis of each goal. This analysis is specified using rationale diagrams.

A rationale diagram appears as a balloon within which goals of a specific actor are analyzed and dependencies with other actors are established. Goals are decomposed into subgoals and positive/negative contributions of subgoals to goals are specified.

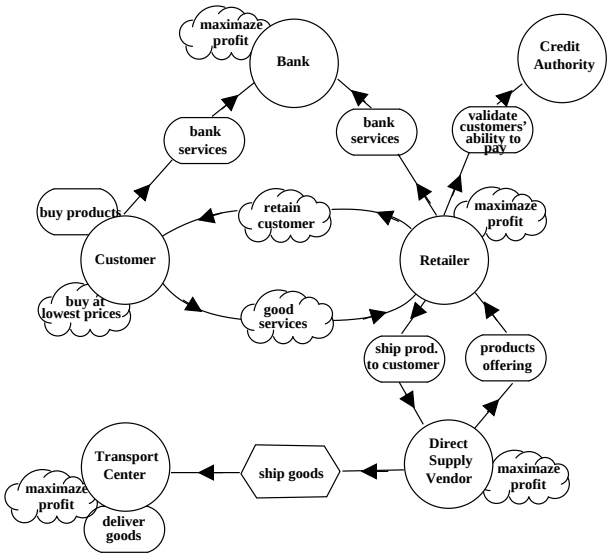

Figure 1: Actor diagram for the online retail store example

Figure 2 shows part of the rationale diagram for Retailer actor. The diagram analyzes to main goal for the retailer: increase return on investment, mainly related to the softgoal maximize profits, and sell products. The goal increase return on investment is AND-decomposed in increase sales volume and increase profit per sale, which are further OR-decomposed. In particular, increase sales volume is decomposed in increase customer appeal and expand market, whereas increase profit per sale is decomposed in increase sale price, lower fix costs, and increase high margin profits. The goal sell products is OR-decomposed in three subgoals which identify three different selling modalities: self serve, auction, and salesperson. Each of these subgoals are further AND-decomposed in subgoals. So for instance, the auction goal is decomposed in cataloging, handle order, and handle request from bidder. In turn these subgoals are further analyzed and decomposed. The diagram shows also contribution links (positive and negative) between goals. For instance, the adoption of the auction selling modality gives a positive contribution to the satisfaction of the goals increase sale price and lower sale price, whereas the modality salesperson gives a negative contribution to lower sale price and a positive contribution to the goal increase customer appeal.

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 3 shows part of the rationale diagram for the retailer actor. In particular, the diagram focus on the dependencies between the retailer and retailer

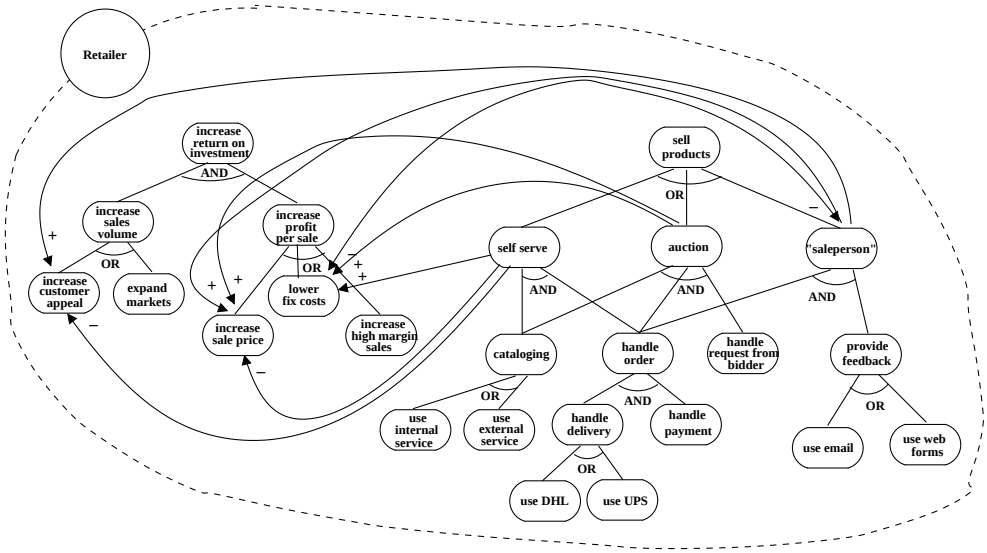

Figure 2: Rationale diagram for the Retailer actor

system actors. Retailer depends on the Retailer System to sell products guaranteeing security, reliability, and performances.

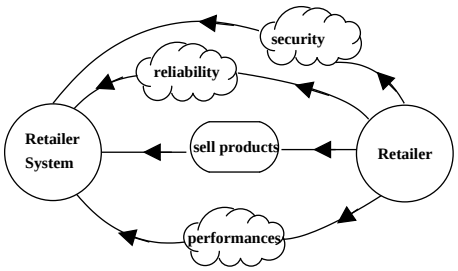

Figure 3: Part of the actor diagram focusing on the functional a non-functional requirements the Retailer actor delegates to the Retailer System

In Figure 4 shows part of the rationale diagram for the retailer system. Basically, this analysis extends the goal analysis done for 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 sake of simplicity, the diagram reports only some of the possible contribution links between services and qualities.

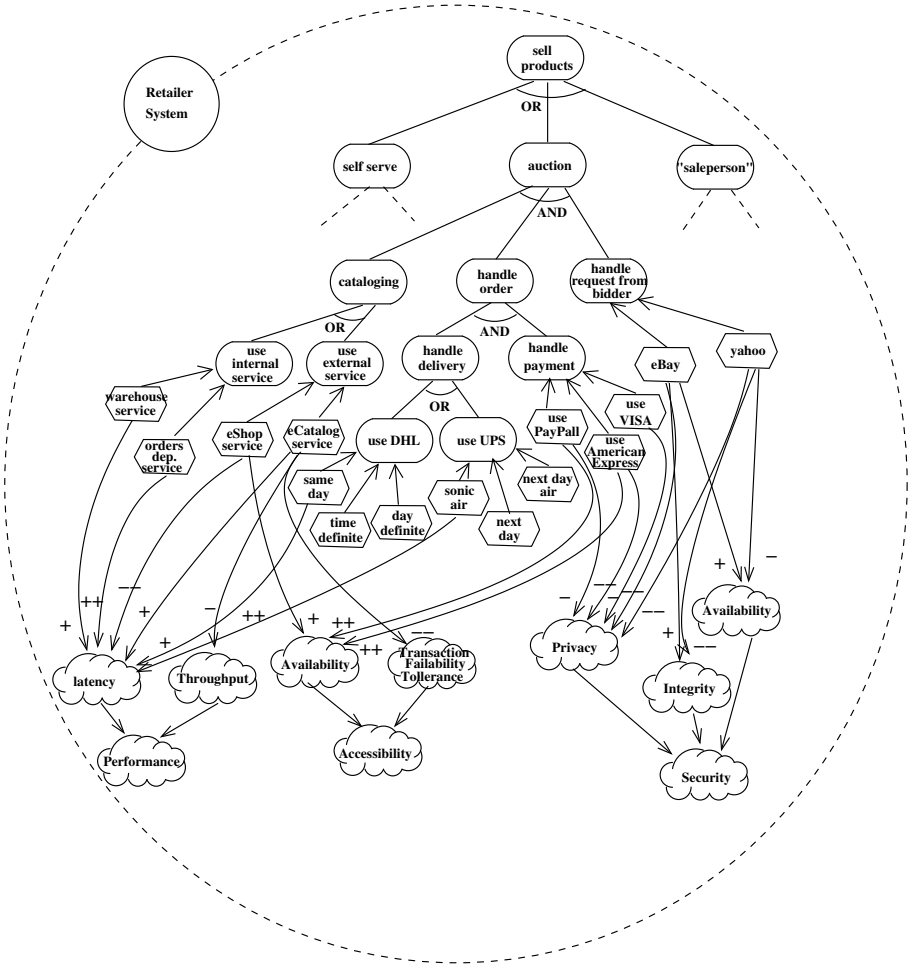

Figure 4: Part of the rationale diagram for the Retailer System.

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 [10, 19].

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 –. These relations have been presented in [10]; here we briefly recall the intuitive meaning.

$G_2 \stackrel{+_S}{\longmapsto} G_1$ [resp. $G_2 \stackrel{++_S}{\longmapsto} G_1$] means that if G_2 is satisfied, then there is some [resp. a full] evidence that G_1 is satisfied, but if G_2 is denied, then nothing is said about the denial of G_1; $G_2 \stackrel{-_S}{\longmapsto} G_1$ [resp. $G_2 \stackrel{--_S}{\longmapsto} G_1$] means that if G_2 is satisfied, then there is some [resp. a full] evidence that G_1 is denied, but if G_2 is denied, then nothing is said about the satisfaction of G_1. The meaning of $+_D$, $-_D$, $++_D$, $--_D$ is dual w.r.t. $+_S$, $-_S$, $++_S$, $--_S$ respectively. (By "dual" we mean that we invert satisfiability with deniability.) The relations +, −, ++, −− are such that each $G_2 \stackrel{r}{\longmapsto} G_1$ is a shorthand for the combination of the two corresponding relationships $G_2 \stackrel{r_S}{\longmapsto} G_1$ and $G_2 \stackrel{r_D}{\longmapsto} G_1$. We call the first kind of relations *symmetric* and the latter two *asymmetric*.

Figure 5 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. For sake of simplicity, we have used only symmetric binary relations and we have reported only the analysis regarding privacy quality. The contribution links between services and privacy quality are both qualitative and quantitative, next we present the difference between the two.

3.1 Qualitative reasoning

Let $G_1, G_2, ...$ denote goal labels. We introduce four distinct predicates over goals, $FS(G)$, $FD(G)$ and $PS(G)$, $PD(G)$, meaning respectively that there is (at least) *full* evidence that goal G is satisfied and that G is denied, and that there is at least *partial* evidence that G is satisfied and that G is denied. We also use the proposition $\top$ to represent the (trivially true) statement that there is at least null evidence that the goal G is satisfied (or denied). Notice that the predicates state that there is *at least* a given level of evidence, because in a goal graph there may be multiple sources of evidence for the satisfaction/denial of a goal. We introduce a total order $FS(G) \geq PS(G) \geq \top$ and $FD(G) \geq PD(G) \geq \top$, with the intended meaning that $x \geq y$ if and only if $x \rightarrow y$. We call FS, PS, FD and PD the possible *values* for a goal.

We want to allow the deduction of *positive* ground assertions of type $FS(G)$,

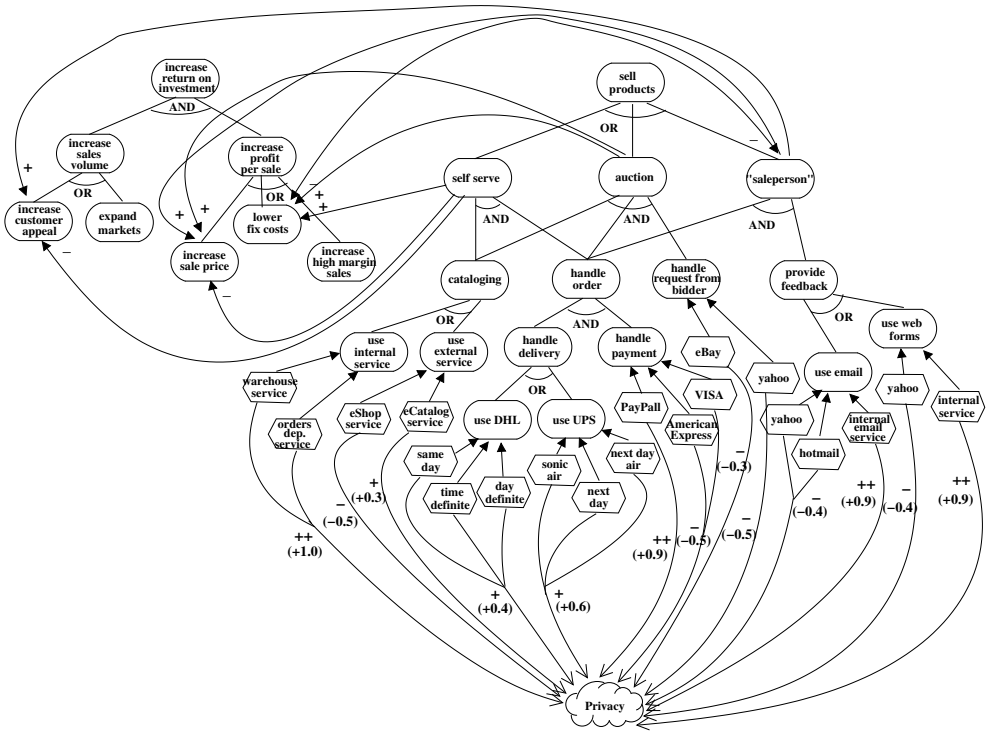

Figure 5: Partial goal model used for the reasoning about qualities

$FD(G)$, $PS(G)$ and $PD(G)$ over the goal constants of a goal graph. We refer to externally provided assertions as *initial conditions*. To formalize the propagation of satisfiability and deniability evidence through a goal graph, we introduce the axioms described in Figure 6.

(1) states that full satisfiability and deniability imply partial satisfiability and deniability respectively. For an AND relation, (2–3) show that the full and partial satisfiability of the target node require respectively the full and partial satisfiability of all the source nodes; for a "$+_S$" relation, (4–7) show that only the partial satisfiability (but not the full satisfiability) propagates through a "$+_S$" relation. Thus, an AND relation propagates the minimum satisfiability value (and the maximum deniability one), while a "$+_S$" relation propagates at most a partial satisfiability value.

We say that an atomic proposition of the form $FS(G)$, $FD(G)$, $PS(G)$ and $PD(G)$ *holds* if either it is an initial condition or it can be deduced via modus ponens from the initial conditions and the ground axioms of Figure 6. Notice that all the formulas in our framework are propositional Horn clauses, so that deciding if a ground assertion holds not only is decidable, but also it can be decided in polynomial time.

Goal	**Invariant Axioms**	
$G:$	$FS(G) \rightarrow PS(G), \quad FD(G) \rightarrow PD(G)$	(1)
Goal relation	**Relation Axioms**	
$(G_1, ..., G_i, ...G_n) \overset{and}{\longmapsto} G:$	$(\bigwedge_i FS(G_i)) \rightarrow FS(G), \quad (\bigwedge_i PS(G_i)) \rightarrow PS(G)$	(2)
	$\bigwedge_i (FD(G_i) \rightarrow FD(G)), \quad \bigwedge_i (PD(G_i) \rightarrow PD(G))$	(3)
$G_2 \overset{+_S}{\longmapsto} G_1:$	$PS(G_2) \rightarrow PS(G_1)$	(4)
$G_2 \overset{-_S}{\longmapsto} G_1:$	$PS(G_2) \rightarrow PD(G_1)$	(5)
$G_2 \overset{++_S}{\longmapsto} G_1:$	$FS(G_2) \rightarrow FS(G_1), \quad PS(G_2) \rightarrow PS(G_1)$	(6)
$G_2 \overset{--_S}{\longmapsto} G_1:$	$FS(G_2) \rightarrow FD(G_1), \quad PS(G_2) \rightarrow PD(G_1)$	(7)

Figure 6: Ground axioms for the invariants and the propagation rules in the qualitative reasoning framework. The (*or*), $(+_D)$, $(-_D)$, $(++_D)$, $(--_D)$ cases are dual w.r.t. (*and*), $(+_S)$, $(-_S)$, $(++_S)$, $(--_S)$ respectively.

Forward propagation. The algorithm and its implementation for this kind of reasoning has been presented in [10]. 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. As the goal graph may be cyclic, the process stops when a fixpoint is reached. The user then can look the final values of the goals of interest (typically root goals), and reveal possible conflicts. The whole algorithm is linear in time as it requires no form of search.

For the example in Figure 5, we could, for instance, be interested in finding

the effects of adopting a set of services over the top goals (i.e., increase return on investment and sell products) as well as over the privacy softgoal (i.e., Security). So for instance, suppose we decide to use the following services (i.e., assigning FS label to them): eShop service, day definite (by DHL), and PayPall. With this configuration we obtain that Privacy is partially satisfied (PS), sell products is fully satisfied (FS), and increase return on investment is partially denied (PD). Note that we have not said anything about the satisfaction of the subgoals of the goal increase return on investment. Of course, we could suppose that expand the markets and then increase return on investment will be also partially satisfied. However, in this case we will have a conflict, that is, we have evidence both for the satisfaction and the denial of the goal.

Goal satisfiability. The implementation of this type of reasoning has been presented in [19]. 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.

For 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. So for instance, suppose we want to fully satisfy sell products and at least partially satisfy Privacy softgoal. The software gives no solution for the fully satisfaction of softgoal Privacy, but it produces several solutions for its partial satisfaction. Fixing the same cost for all services, one of this solution consists of the following services: same day (by DHL), PayPall, and internal email service.

3.2 Quantitative reasoning

The qualitative approach allows for setting and propagating partial evidence about the satisfiability and deniability of goals and the discovery of conflicts.

We may want to provide a more fine-grained evaluation of such partial evidence. For instance, when we have $G_2 \xmapsto{+_S} G_1$, from $PS(G_2)$ we can deduce $PS(G_1)$, whilst one may argue that the satisfiability of G_1 is in some way less evident than that of G_2. For example, in the goal model of Figure 5, the use of the service PayPall may not necessarily imply satisfaction of Privacy softgoal, so it may be reasonable to assume "less evidence" for the satisfaction of the latter compared to the former. Moreover, the different relations which mean partial support – i.e., $+_S$, $-_S$, $+_D$, $-_D$ – may have different strengths. For instance, in our example the sonic air service may have a bigger impact on Privacy than eShop service.

To cope with these facts, we need a way for representing different *numerical* values of partial evidence for satisfiability/deniability and for attributing different weights to the $+_S$, $-_S$, $+_D$, $-_D$ relations. the formal framework to reason with such quantitative information has been presented in [10, 7]. We recall in the following briefly the main features.

We introduce two real constants inf and sup such that $0 \leq inf < sup$. For

each node $G \in \mathcal{G}$ we introduce two *real* variables $Sat(G), Den(G)$ ranging in the interval $\mathcal{D} =_{def} [inf, sup]$, representing the current evidence of satisfiability and deniability of the goal G. The intended meaning is that inf represents no evidence, sup represents full evidence, and different values in $]inf, sup[$ represent different levels of partial evidence.

To handle the goal relations we introduce two operators $\otimes, \oplus : \mathcal{D} \times \mathcal{D} \longmapsto \mathcal{D}$ representing respectively the evidence of satisfiability of the conjunction and that of the disjunction [deniability of the disjunction and that of the conjunction] of two goals. $\otimes$ and $\oplus$ are associative, commutative and monotonic, and such that $x \otimes y \leq x, y \leq x \oplus y$; there is also an implicit unary operator $inv()$, representing negation, such that $inf = inv(sup)$, $sup = inv(inf)$, $inv(x \oplus y) = inv(x) \otimes inv(y)$ and $inv(x \otimes y) = inv(x) \oplus inv(y)$.

We also attribute to each goal relation $+_S$, $-_S$, $+_D$, $-_D$ a weight $w \in]inf, sup[$ stating the strength by which the satisfiability/deniability of the source goal influences the satisfiability/deniability of the target goal. As in the qualitative approach, we use the symmetric relation —such as, $G_2 \overset{w+}{\longmapsto} G_1$— as a shorthand for the combination of the two corresponding asymmetric relationships sharing the same weight w —e.g., $G_2 \overset{w+_S}{\longmapsto} G_1$ and $G_2 \overset{w+_D}{\longmapsto} G_1$.

There are a few possible models following the schema described above. In particular, here we adopt a *probabilistic* model, where the evidence of satisfiability $Sat(G)$ [resp. deniability $Den(G)$] of G is represented as the probability that G is satisfied (respectively denied). As usual, we adopt the simplifying hypothesis that the different sources of evidence are independent. Thus, we fix $inf = 0$, $sup = 1$, and we define $\otimes, \oplus, inv()$ as:

$$p_1 \otimes p_2 =_{def} p_1 \cdot p_2, \quad p_1 \oplus p_2 =_{def} p_1 + p_2 - p_1 \cdot p_2, \quad inv(p_1) = 1 - p_1$$

that is, respectively the probability of the conjunction and disjunction of two independent events of probability p_1 and p_2, and that of the negation of the first event. To this end, the propagation rules are those of a Bayesian network, where, e.g., in $G_2 \overset{w+_S}{\longmapsto} G_1$ w has to be interpreted as the conditional probability $P[G_1\ is\ satisfied \mid G_2\ is\ satisfied]$.

As with the qualitative case, we call a *value statement* an expression of the form $(v \geq c)$, $v \in \{Sat(G_i), Den(G_i)\}$ for some G_i and $c \in [0, 1]$, with the intuitive meaning "there is at least evidence c of v". We want to allow the user to state and deduce non-negated value statements of the kind $(Sat(G) \geq c)$ and $(Den(G) \geq c)$ over the goal constants of the graph. As before, we call externally provided assertions about the satisfaction/denial of goals *initial conditions*.

To formalize the propagation of satisfiability and deniability evidence values through a goal graph, for every goal and goal relation, we introduce the axioms (8)-(15) in Figure 7. Unlike those of Figure 6, the relation axioms in Figure 7 are not ground Horn clauses —thus, propositional— but rather first-order closed Horn formulas, so that they require a first-order deduction engine.

We say that a statement $(v \geq c)$ holds if either it is an initial condition or it can be deduced from the initial conditions and the axioms of Figure 7.

Goal relation	Axioms	
$(G_2, G_3) \overset{and}{\longmapsto} G_1$:	$(Sat(G_2) \geq x \wedge Sat(G_3) \geq y) \rightarrow Sat(G_1) \geq (x \otimes y)$	(8)
	$(Den(G_2) \geq x \wedge Den(G_3) \geq y) \rightarrow Den(G_1) \geq (x \oplus y)$	(9)
$(G_2, G_3) \overset{or}{\longmapsto} G_1$:	$(Sat(G_2) \geq x \wedge Sat(G_3) \geq y) \rightarrow Sat(G_1) \geq (x \oplus y)$	(10)
	$(Den(G_2) \geq x \wedge Den(G_3) \geq y) \rightarrow Den(G_1) \geq (x \otimes y)$	(11)
$G_2 \overset{w+_S}{\longmapsto} G_1$:	$Sat(G_2) \geq x \rightarrow Sat(G_1) \geq (x \otimes w)$	(12)
$G_2 \overset{w-_S}{\longmapsto} G_1$:	$Sat(G_2) \geq x \rightarrow Den(G_1) \geq (x \otimes w)$	(13)
$G_2 \overset{++_S}{\longmapsto} G_1$:	$Sat(G_2) \geq x \rightarrow Sat(G_1) \geq x$	(14)
$G_2 \overset{--_S}{\longmapsto} G_1$:	$Sat(G_2) \geq x \rightarrow Den(G_1) \geq x$	(15)
$G_2 \overset{w+_D}{\longmapsto} G_1$:	$Den(G_2) \geq x \rightarrow Den(G_1) \geq (x \otimes w)$	(16)
$G_2 \overset{w-_D}{\longmapsto} G_1$:	$Den(G_2) \geq x \rightarrow Sat(G_1) \geq (x \otimes w)$	(17)
$G_2 \overset{++_D}{\longmapsto} G_1$:	$Den(G_2) \geq x \rightarrow Den(G_1) \geq x$	(18)
$G_2 \overset{--_D}{\longmapsto} G_1$:	$Den(G_2) \geq x \rightarrow Sat(G_1) \geq x$	(19)

Figure 7: Axioms for the propagation rules in the quantitative reasoning framework.

We implicitly assume that $(Sat(G_i) \geq 0)$ and $(Den(G_i) \geq 0)$ hold for every G_i, and that the deduction engine —either human or machine— can semantically evaluate $\otimes$ and $\oplus$ and perform deductions deriving from the values of the evaluated terms and the semantics of $\geq$. For instance, we assume that, if $(G_2, G_3) \overset{and}{\longmapsto} G_1$ is in $\mathcal{R}$, then $(Sat(G_1) \geq 0.1)$ can be deduced from $(Sat(G_2) \geq 0.5)$, $(Sat(G_3) \geq 0.4)$, as from (8) it is deduced $(Sat(G_1) \geq 0.5 \otimes 0.4)$, which is evaluated into $(Sat(G_1) \geq 0.2)$, from which it can be deduced $(Sat(G_1) \geq 0.1)$.

Forward reasoning. As for the qualitative reasoning here the problem is the same with only difference that now we use probability values as initial assignment. The algorithm and its implementation has been presented in [10].

Suppose in our example, we want to use the services eShop service, day definite (by DHL), and PayPall (i.e., we assign them 1.0 as initial satisfaction value). As consequence we obtain that Privacy assumes S=0.4 and D=0.5 and sell products S=1.0, that is we can satisfy our top goal but we cannot say much about Privacy. Note that we have not considered the effects produced by the adopted services on the increase return on investment goal.

Goal satisfiability. The solution and the tool for the quantitative goal satisfiability problem has been presented in [7]. The tool is based on a commercial optimization tool, Lingo 8.0 [1] and is integrated with our goal analysis framework.

Analogously to the qualitative case, in our example, we might be interested

[1] http://www.lindo.com

in finding the minimal cost set of services that satisfy totally the top goal sell products (S=1.0) and partially the Privacy softgoals, let's say for example S=0.7. The software produces a set of solutions that satisfy such a request, i.e., a set of services that, if adopted, produce the desiderata 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 measure for software and, in particular, for services, but are not the only possible ones. Availability, integrity, regulatory conformance are other QoS that may need to be modeled, furthermore, other parameters specific to the application at hand may need modeling. 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 describe their qualities and have shared vocabulary of service qualities. Standard service description languages such as WSDL 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 [11]); on the other hand, one could complement WSDL interfaces with ancillary documentation for the description of quality of service characteristics of the service.

In [17] a symmetric model based on constraint satisfaction techniques is used to verify QoS desires coming from the requester. In [21], a XML based 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 [20]. A predictive QoS model for workflows involving QoS properties is proposed in [3]. 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, [15]. What is missing is a framework to design composition and orchestrations of services with desired global QoS requirements and, dually, to analyze 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:SecureSocketLayerVersion3.1</wsse:TokenType>
05      </wsse:SecurityToken>
06      <wsse:SecurityToken wsp:Usage="wsp:Required" wsp:Preference="60">
07
<wsse:TokenType>wsse:SecureSocketLayerVersion3.0</wsse:TokenType>
08      </wsse:SecurityToken>
09      <wsse:SecurityToken wsp:Usage="wsp:Required" wsp:Preference="10">
10
<wsse:TokenType>wsse:SecureSocketLayerVersion2.0</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 `SecurityToken`s. 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 modeling 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 satisfiability have be proposed to design an architecture meeting given QoS requirements, but also to understand which QoS properties will a system have and how the various services influence the QoS parameters. We have based our investigation on a representative case study and have shown that a wide range of non-functional properties can 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 limitation of the proposed approach is that the contribution of the single service to a given element of the quality of service set 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.

Acknowledgments We would like to thank John Mylopoulos, Mike Papazoglou, and Roberto Sebastiani for useful comments, discussions and feedback.

References

[1] V. D. Andrea and M. Aiello. Services and objects: Open issues. In G. Piccinelli and S. Weerawarana, editors, *European workshop on OO and Web Service*, pages 23–29, 2003. IBM Research Report. IBM. Computer Science, (RA 220).

[2] 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.

[3] 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.

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

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

[6] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.

[7] 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.

[8] 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).

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

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

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

[12] 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).

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

[14] 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.

[15] 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.

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] E. Yu. Modeling organizations for information systems requirements engineering. In *Proc. of the 1st Int. Symposium on Requirements Engineering, RE'93*, pages 34–41, San Jose, USA, Jan. 1993.

[23] E. Yu. Understanding 'why' in software process modeling, analysis and design. In *Proc. of the 16th Int. Conf. on Software Engineering, ICSE'94*, pages 159–168, Sorrento, Italy, May 1994.

[24] E. Yu. *Modelling Strategic Relationships for Process Reengineering.* PhD thesis, University of Toronto, Department of Computer Science, 1995.

[25] E. Yu and J. Mylopoulos. Using goals, rules, and methods to support reasoning in business process reengineering. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 5(1):1–13, 1996.