

From Stakeholder Needs to Service Requirements

Loris Penserini, Anna Perini, Angelo Susi
ITC-IRST
Via Sommarive 18
I-38050, Trento, Italy
{penserini,perini,susi}@itc.it

John Mylopoulos
University of Trento
Via Sommarive, 14
I-38050, Trento, Italy
jm@cs.toronto.edu

Abstract

Web services offer a novel open, distributed computing environment. Within that environment, web services need to be discovered and composed in order to meet user needs (or, goals). For example, a user goal may be “plan a holiday”. To fulfill this goal, several services need to be discovered from different directories, such as “book hotel”, “book flight”, “rent car” and “pay by credit card”. These then need to be composed into a plan (“process”). In this work, we adopt and extend an agent-oriented software design methodology (Tropos) and tailor it to the design of web services. We also sketch how Tropos design-time models can support service discovery and composition by offering a roadmap that relates stakeholder goals to collections of services available in different directories. A case study from the literature is used throughout.

1. Introduction

Web services promise to revolutionize software, and for that matter Software Engineering. But the research that will realize this revolution is only beginning.

Web services are to be discovered and composed at run-time. For this to be possible, they need to be wrapped with their associated “semantics”, to facilitate discovery and support composition. What are those “semantics”? It seems sensible to assume that this semantics can be derived from the requirements models and design specifications of a web service. But what part of these models will actually be useful at run-time, for discovery and composition purposes? And what is missing, to be provided as a supplement to design-time information? Moreover, what exactly does this semantics buy you in facilitating service discovery and composition from a flat service repository?

We are conducting research on these questions. To start off, we need to choose a software development paradigm, which determines the nature of requirements and designs.

The potential contribution from the Multi Agent Systems (MAS) research areas, in terms of conceptual framework and technologies has been pointed out by several authors, see for instance [23]. Moreover, like other researchers, we anticipate that for web services to succeed, they will have to be generic software systems that can be customized in a variety of ways to meet different operating environments, user preferences and skills. In addition, they will need to have social skills to support negotiation, delegation and monitoring, so that they can cope with run-time emergencies. In this work we adopt an agent-oriented software development methodology, named *Tropos* [5, 6], and study the questions posed earlier in the context of *Tropos* models for requirements and design. The main objective of this work is to refine the *Tropos* methodology in order to support web service design. Specifically, we extend the *Tropos* design process to support a revised notion of capability which explicitly correlates actor plans with stakeholder needs and environmental constraints. We also show how design-time models can serve as a roadmap for selecting collections of services which collectively can fulfill user goals.

The rest of the paper is structured as follows. Section 2 reviews basic features of the *Tropos* methodology and recalls previous research on *Tropos* for web service design. Section 3 presents our extension of *Tropos* concepts and a corresponding refinement of the design process. The process is illustrated with a simplified example taken from a web-based online selling scenario. This example is then described in more details to illustrate how to build a services model (Section 4) and how to use this services model (Section 5). Section 6 describes related works and finally Section 7 presents concluding remarks.

2. Background

We focus here on the *Tropos* Agent-Oriented methodology [5, 6] which bears particular attention to stakeholder needs and requirements analysis. More precisely, the *Tropos* methodology guides the software engineer in building a

series of models, which are incrementally refined and extended, from an early requirements model to system design artifacts and then to code. The methodology provides a modelling language based on a multi-agent paradigm named *i** [24], various analysis techniques and a structured modelling (design) process supported by a set of tools [20].

Basic constructs of the conceptual modelling language are those of *actor*, *goal*, *plan*, *softgoal*, *resource* and *capability*. Dependency links between pairs of actors allow to model the fact that one actor depends on another in order to achieve a goal, execute a plan, or acquire a resource. This type of information can be graphically depicted in an actor diagram, a graph whose nodes represent actors (circles) and whose arcs represent dependencies (a couple of arrows linked by its dependum), as illustrated in Fig. 7. Goal analysis techniques support AND/OR goal decomposition of goals, means-ends analysis of plans and resources, and contribution analysis to goals and softgoals that aggregate all contributions –positive or negative– to a goal to determine the degree to which it is fulfilled. Goal analysis from the owner actor perspective is visualized via a sort of balloon capturing the goals tree decomposition, means-ends relationships and contribution relationships (an example is in Fig. 8).

The software development process in *Tropos* is organized into five phases, each characterized by specific objectives. In particular, during *Early Requirements* the environment (i.e. the organizational setting) is modelled and analyzed; during *Late Requirements*, the system-to-be is introduced and its role in fulfilling stakeholder goals is determined; during *Architectural Design* the system architecture is specified in terms of a set of interacting software agents; *Detailed Design* is concerned with the specification of software agent capabilities and interactions; finally, during *Implementation* code is generated from the detailed design specification. The modelling process which is iterated along the four design phases has been described in [5] in terms of a non-deterministic concurrent process and is recalled and extended in Section 3.

The rest of the section is devoted to recall approaches that exploit the *Tropos* methodology for the design and run-time support of generic services and web services. A requirement-driven approach to web service development that takes into account global business rules (i.e. rules of engagement between different organizations) and local business rules (i.e. rules deriving from the business needs of each local partner) is proposed in [21]. Here the *Tropos* methodology is exploited to model choreography (global) and orchestration (local) requirements. In particular, *Tropos* actor diagrams are used to model global strategic requirements, in terms of actor goals and dependencies. *Tropos* goal diagrams are used to model single actor needs. Goal refinement from the point of view of each single actor leads

to leaf goals which are further specified along with their procedural requirements using UML activity diagrams. Activity diagrams can then be transformed into executable code (BPEL4WS [10]). Model checking techniques (Formal Tropos [8, 12]) are exploited to verify valid behaviors of the *Tropos* model which specify the service requirements allowing to prevent problems related to a wrong or under-specified requirements definition.

[13] also proposes a *Tropos*-based web service design methodology. Here early and late requirements are used to specify web service functionalities and the requirements of integration with existing software systems; also to provide the reasons for a service being offered. In the two design phases (architectural and detailed design) a *Tropos* specification is combined with web service specification technologies, such as WSDL [1] and BPEL4WS. In particular, *Tropos* means-ends relationships between tasks and goals are related to business processes, agent capabilities modeled at the detailed design stage correspond to WSDL specifications, and agent interaction modelling is replaced by BPEL process specifications.

More emphasis on the impact of design level analysis at run time is given in [3] and in [15]. The first work proposes to adapt and use the *Tropos* goal analysis techniques to model and reason on Quality of Service (QoS) requirements, that is on non-functional properties of a software service such as performance, cost, reliability and security.

The second work introduces a new concept, called the initial service need (*isn*), with the aim to model the reason for a service request, which can be derived from a model of the consumer needs. An *isn* can be used at run-time to enhance mapping mechanisms exploited, for instance, during service discovery.

Our work refines some of the ideas proposed in the above mentioned research and defines novel extensions to the *Tropos* design process together with tools that support it [18].

3. The approach

This section will first illustrate the basic concepts we use for modelling stakeholder needs and system services. In particular, it will give a revised notion of capability, with respect to the one defined in [5] and used for service detailed design in [13]. It will then describe the extensions to the design process algorithm which is iterated along the software development process, from early requirements to detailed design, that are needed in order to integrate those refined concepts.

3.1. Concepts definition

The *Tropos* methodology provides concepts and analysis techniques to model stakeholder intentions in terms of goal

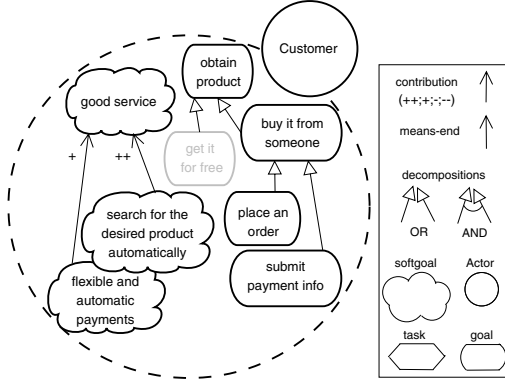


Figure 1. Modeling stakeholder needs in Tropos.

and plan relationships—they have been briefly recalled in Section 2. These methods and techniques can be straightforwardly applied to identify and model the need of a service by a given stakeholder (consumer) in a particular application domain. Consumers will be modeled as actors and their needs will be modeled in terms of goals, eventually connoted by softgoals. An example, referring to a web-based online selling scenario, is given in Fig. 1 where a customer is modeled as a *Tropos* actor (a circle) and its need of obtaining a product by accessing a good service is analyzed in terms of goal decompositions and softgoal contributions¹.

Focusing on service design, in order to explicitly link the identified consumers’ needs to a service specification we propose to model services in terms of system capabilities, since the requirements analysis phases, and extend the definition of this concept following recent works on agent paradigms. Specifically, according to [17], the notion of capability should refer to the philosophical idea that ‘can’ implies both *ability* and *opportunity*, so that the lack of either ability or opportunity implies ‘cannot’. In other words, the meaning of the concept of “capability for a given goal” is that the agent has at least one plan—the ability—that can be triggered by the goal. This defines a necessary condition for achieving the goal, while the sufficient condition (i.e. the opportunity) is defined in terms of the pre-conditions or the context that can trigger the plan.

Turning to our methodology, we propose to describe the ability part via the *Tropos* means-ends relationship between a system goal and a plan, while the opportunity part will be described via system plan/softgoal contributions, $\langle plan, softgoal, metric \rangle$ ($metric \in \{-, -, +, ++\}$) and via environmental constraints (e.g. temporal constraints between subplans) that will be specified by model annota-

¹Notice that, for softgoal contributions we adopt qualitative relationships which represent full contribution (‘++’, ‘-’) and the partial contribution (‘+’, ‘-’). More complex relationships can be used as discussed in [9].

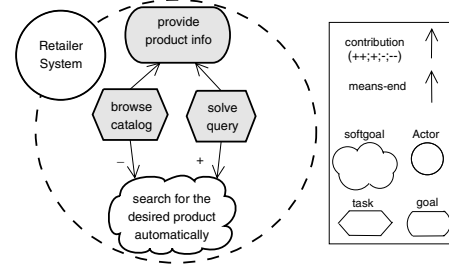


Figure 2. Example of the structure of two Tropos capabilities.

tions. More formally, the definition of capability (*cap*) is given as:

$$cap = \langle means_end(goal, plan), \\ \cup_i contribution(plan, softgoal_i, metric), \\ \{stakeholder_needs \text{ and } domain_constraints\} \rangle$$

where $contribution(plan, softgoal_i, metric)$ is the set of contribution relationships of the plan *plan* to the softgoals $softgoal_i$, according to the chosen contribution *metric*.

Fig. 2 shows elements of a capability model, namely, Cap_1 , which rests on the means-ends relationship *provide_product_info*, *browse_catalog*, and Cap_2 , which rests on the means-ends relationship *provide_product_info*, *solve_query*. Their *opportunity* part is represented by the $+/-$ contributions to the softgoal *search for the desired product automatically*. Notice that this softgoal, appeared first in our customer’s need (Fig. 1), and has been delegated to the system actor during the analysis, as explained in Section 4. Both capabilities meet the same system goal (e.g. *provide product info*), so they model alternative ways to achieve it, even if with different strength. A capability model will be enriched with process (i.e. dynamic) properties at detailed design [17], in order to provide an executable service specification.

Alternative capabilities (services) can be traced back to a given consumer’s need at design time, by exploiting the concept of initial service need, that was originally introduced in [15] for semantic web services. An initial service need (*isn*) is composed of a set of goals (*g*) and softgoals (*sg*), as follows:

$$isn = \langle (g_1, sg_{1,1}, \dots, sg_{1,n}), \dots, (g_m, sg_{m,1}, \dots, sg_{m,k}) \rangle$$

As clarified by the examples in next sections, each *isn* drives the analyst, at design time, in finding the more convenient capability, among the modeled alternatives, i.e. each *isn* defines a solution-tree (a hierarchy of goals, and plans along with their relations). Thanks to this concept, we are able to build up a roadmap that relates user goals to collections of services. Moreover, this concept can be used also

at run time (instance-level) when changes in consumers' needs or instantiations of environmental constraints may disable the execution of a given alternative (service) in a non-deterministic way. Hence, a wise choice at design-time could be to define different plausible alternatives in terms of specific system behaviors that the system can separately adopt to cope with contingencies, e.g. failure handling related to monitoring and compensation activities.

3.2. The Design Process

The *Tropos* design process, given in [5], specifies in terms of a non deterministic concurrent algorithm the steps that will be carried out by analysts at design-time. This algorithm has been implemented in a tool which supports the design process by giving suggestions to the user (the analyst) on the possible steps to be performed when building a model². Fig. 3 sketches the original algorithm, given in [5], and points out extensions. During *Phase 1* —i.e. the *initialization* of the process— the set of domain actors and their associated goals is added to the model. During *Phase 2* —i.e. the *modelling* activity process— the *goal_Analysis* procedure is called concurrently for every goal in the model. As shown in Fig. 4, four possible decisions can be taken by the designer with respect to each goal. The *delegate* decision choice can be selected in two cases: when the goal can be delegated to another actor already existing in the model and when the goal can be delegated to actors that have not yet been specified. In this second case, a new actor will be added to the model and the goal is delegated to him/her. Upon the *expand* choice, the goal can be AND/OR decomposed into subgoals; the new subgoals are added to a list containing the goals that are to be further analyzed. This will result in specifying a hierarchy of goals. The *contribute* choice is taken when contribution relationships between the goal *g* to other goals and softgoals have been identified. By choosing *solve*, the designer specifies that the actor is able to satisfy the goal by him/herself. Notice that the *model.add* procedure abstracts possible modeling updates that the analyst will undertake. The process is complete when all goals are satisfied. This process follows a top-down approach driven by the *expand* step in the goal analysis that will allow to refine abstract goals into more specific ones.

A first extension of the design process consists in allowing for bottom-up analysis. A second one in introducing a capability modelling procedures, as detailed in the following.

Phase 1 of the procedure —i.e. *initialization* process— has been modified in order to add a step devoted to plans specification (bold lines in Fig. 3), such a step is carried out independently and in parallel with the collection of actors

²The description of the tool is not included here, due to lack of space. More details on this can be found in [18].

Generate_Tropos_Model(model)

Phase 1: initialization

collect the set of domain actors *A*, goals *G* softgoals *SG*,
plans *P*, *model.add(A,G,SG,P)*;

For every goal *g*, softgoal *sg*, plan *p*

assign *g* and *sg* to an actor *a*;

assign *p* to an actor *a*;

Phase 2: modelling

Concurrently For every actor *a*

Concurrently For every goal *g* and **plan *p***

goal_analysis(g);

plan_analysis(p);

Verify goals satisfaction

Figure 3. The algorithm of the Tropos design process as published in [5] and its refinements (lines in bold)

procedure goal_analysis(g)

decision = decision_on_goal(g);

case decision:

- *delegate g* to existing actors or new actors $\{a_i\}$,
model.add($\{a_i\}$),
model.add(*dependency*(*a_depender*, *g*, *a_dependee*));
- *expand g* in subgoals $\{g_i\}$,
model.add($\{g_i\}$),
model.add(*decomposition_{and/or}*(*g*, $\{g_i\}$));
- *contribute g/sg* to goals/softgoals $\{g_i, sg_i\}$,
model.add(*contribution*(*g/sg*, $\{g_i/sg_i\}$, *metric*));
- *solve* associating *g* to a plan, ***capability_modeling(g)***;

Figure 4. The goal analysis procedure in the Tropos process (in bold the refinement of the capability modelling).

and goals. *Phase 2* of the process has been modified introducing a plan modelling step, briefly sketched in Fig. 5, where the new *plan_analysis* procedure is shown. During plan modelling the possible decisions can be: *delegate* the plan to other, eventually new, actors; *expand* a plan into sub-plans; *contribute*, i.e. explicit contributions of the plan *p* to some goals; or *compose* the plan with other plans executable by the actor into new higher level plans. This procedure generates plan hierarchies that enable the description of different alternatives for the fulfillment of a certain goal. Notice also that the *compose* choice implement a bottom-up problem solving strategy.

Now the design process can proceed in two directions, eventually in parallel: a top-down analysis that refines set of higher level goals/plans into more specific ones; a bottom-up analysis that enables the composition of plans into higher

```

procedure plan_analysis(p)
decision = decision_on_plan(p);
case decision:
- delegate p to existing actors or new actors  $\{a_i\}$ ,
  model.add( $\{a_i\}$ ),
  model.add(dependency(a_depender, g, a_dependee));
- expand p in sub-plans  $\{p_i\}$ ,
  model.add( $\{p_i\}$ ),
  model.add(decomposition_and_or(p,  $\{p_i\}$ ));
- compose p with a set of plans  $\{p_i\}$ 
  into a new plan p0, model.add(p0),
  model.add(decomposition_and_or(p0,  $\{p_i\}$ ));
- contribute p to goals/softgoals  $\{g_i, sg_i\}$ ,
  model.add(contribution(p,  $\{g_i/sg_i\}$ , metric));
- solve associating p to a goal, capability_modelling(p);

```

Figure 5. The plan analysis procedure in the refined Tropos process.

```

procedure capability_modelling(entity)
if entity is a goal g
  For every plan p
    if p solve g then model.add(means_end(g, p))
    model.add(constraints_from_the_domain)
    CapabilityTable.add(capabilityName,
      (means_end(g, p),  $\cup_i$  contribution(p, sg, metric),
      {constraints_from_the_domain}))
else if entity is a plan p
  For every goal g
    if p solve g then model.add(means_end(g, p))
    model.add(constraints_from_the_domain)
    CapabilityTable.add(capabilityName,
      (means_end(g, p),  $\cup_i$  contribution(p, sg, metric),
      {constraints_from_the_domain}))

```

Figure 6. An outline of the process for analyzing capabilities.

level plans that become means to *solve* goals. Moreover, the possibility to specify means-ends relationship between a goal and a set of plans provides a mechanism for the specification of different, alternative ways to fulfill the goal. The *capability modeling procedure* rests on this basic mechanism.

The correspondent algorithm is given in Fig. 6. Specifically, for each goal, the procedure allows to discover plans that can fulfill the goal and it adds a means-ends relationship for every discovered goal/plan pair; these pairs will constitute the first part of the definition of a capability. For every discovered means-ends, the algorithm collects the set of softgoal contribution relationships related to the plan;

these contributions represent conditions from the domain for the capability, i.e. fragments of its opportunity part. Up to now, we distinguished between two principal kinds of softgoals contributions: those related to domain constraints—e.g. softgoals that model pre- and post- conditions from the environment— and those related to stakeholders needs—e.g. preferences that are modelled as set of goals and softgoals coming from inter-actor dependencies and contribution links analysis. Therefore, at design time, the output of the capability modelling step is relevant³ in order to effectively deal with correlation between capabilities and stakeholders needs. Indeed, the outcome of such an algorithm is double: first, it provides a list of all the capabilities along with their contribution to softgoals achievement, and, second, it groups such capabilities according to their total (positive, negative) contribution to softgoals which are part of stakeholders needs.

This design process, including Agent Oriented modelling and code generation, is supported by the *TAOM4E* tool⁴ which exploits automatic transformation techniques as described in [16, 17].

4. Building a service specification

The example presented in this section is taken from a web based online selling scenario. Its purpose is to illustrate how to build service requirements specification, following the proposed process algorithm (*Phase 1*), in which the analyst assigns goals and plans to specific actors. The actor diagram of Fig. 7, shows five principal domain stakeholders that have been identified at the *early requirements* stage, namely, Retailer, Customer, Local Supplier, UPS Centre, and Bank. Stakeholders' needs are characterized on the basis of their interests, commitments, and responsibilities. For example, Retailer's main objective is to satisfy the softgoal maximize profit, and the principal way to deliver on such an objective is to **sell products**—a root goal for such an actor. Again, the top-level goals for Customer are **obtain products** along with a **good service**. Notice that, a softgoal expresses a preferred condition (e.g. a system non-functional requirement) to be pursued, but stakeholders can actually rely just on basic solutions modelled as goals and plans dependencies.

Fig. 8 shows a fragment of the model refined upon the execution of the *Phase 2* of the design process where several *expand* and *solve* steps, of the procedure *goal_analysis*(*g*), have been performed.

Focusing on the Retailer System actor which represents

³Notice that *CapabilityTable* is not a *Tropos* concept. This structure is kept updated during the design process in order to efficiently support model querying like, for a given system actor, what are its capabilities, etc..

⁴See <http://sra.itec.it/tools/taom4e>.

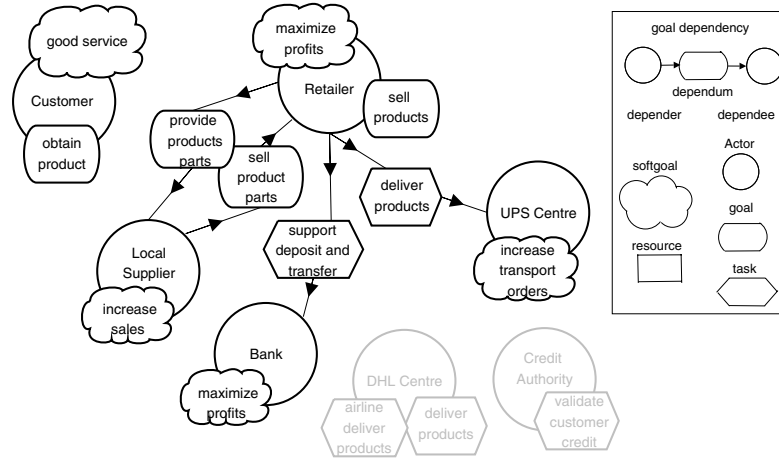


Figure 7. Stakeholder intentions and social dependencies from the early requirements analysis.

an online virtual shop, its services, modeled in terms of system capabilities, are related, through goal dependency, to the following goals: **sell products** maximizing profits of the **Retailer** and **place an order**, by the **Customer**, respectively. But while the **Customer's** goal **place an order** can be linked to a basic system capability, the satisfaction of the **Retailer's** goal **sell products** goes along a more complex analysis of system's goals.

A first outcome of the top-down analysis, when applying the procedure *capability_modelling(g)* of the *Tropos* process algorithm, is illustrated in Tab. 1 that lists all the capabilities that the system-to-be should provide to better accommodate with stakeholders needs. Each capability is characterized by two basic features. The *List of Contributions* which collects positive and negative contributions that will be used to compute a measure of a need coverage. The *Type* feature which can take the following values: *not dependent*, to indicate a system internal capability, for instance, *Cap₁* and *Cap₂* in gray color in Fig. 8; *dependent on*, to indicate that this system capability requires an external capability, e.g. to deal with the goal **handling bank transfers** (part of *Cap₄*), **Retailer System** depends on **Bank** by the plan **support deposit and transfer**; “??” when the type of capability is still to be defined.

Fig. 8 shows the model resulting upon a first top-down analysis. Some of the **Retailer System** functional requirements are not yet covered, e.g., the leave-goals **handling credit card operations** and **cope with worldwide delivery** cannot be internally satisfied, even if, they can be relevant to contribute to some softgoals achievement.

In such a case, a bottom-up approach may be required in order to support at design-time the analyst with new knowledge from the application domain. Notice that, such acquaintances may also come from the run-time environment, e.g., in a MAS scenario, the agents of the run-time environ-

ment may enrich the system-to-be domain knowledge interacting with new partners in a peer-to-peer computing fashion. Specifically, in order to deal with a bottom-up analysis, the analyst follows the *Phase 1* by making explicit domain stakeholders that were not taken into consideration in the previous analysis, e.g.: the actor **DHL Centre** with its plans **airline deliver products** and **deliver products**, and the actor **Credit Authority** with plan **validate customer credit** (they have been drawn by a low opacity line in Fig. 7).

Iterating *Phase 2* and a *plan_analysis(g)*, the analyst can try to check if the new available plans satisfy the **Retailer System** leaf goals, still unsatisfied. The bottom-up analysis results are illustrated in Fig. 9, namely, the system-to-be can be enriched by adding new (external) capabilities: *Cap₃* and *Cap₅*. A further step, left to the software engineer, consists in classifying system capabilities as *publicized*, that is, services that will be automatically detectable via UDDI, or *not-publicized*.

5. Using a service model

Given a service model, we can define several *isn* along with a way to query the model, with the aim to evaluate the coverage of consumers needs by alternative services that have been specified. This offers a roadmap for selecting sets of services, which collectively can fulfill user goals.

We will discuss a couple of examples on the model illustrated in Fig. 8, where three main types of services have been identified during the **Retailer System** analysis:

- *not dependent* and *publicized* web services such as those addressed by *Cap₁* and *Cap₂*;
- *not-publicized* services that the system may also delegate to others actors, e.g. those related to *Cap₃* and *Cap₄*;

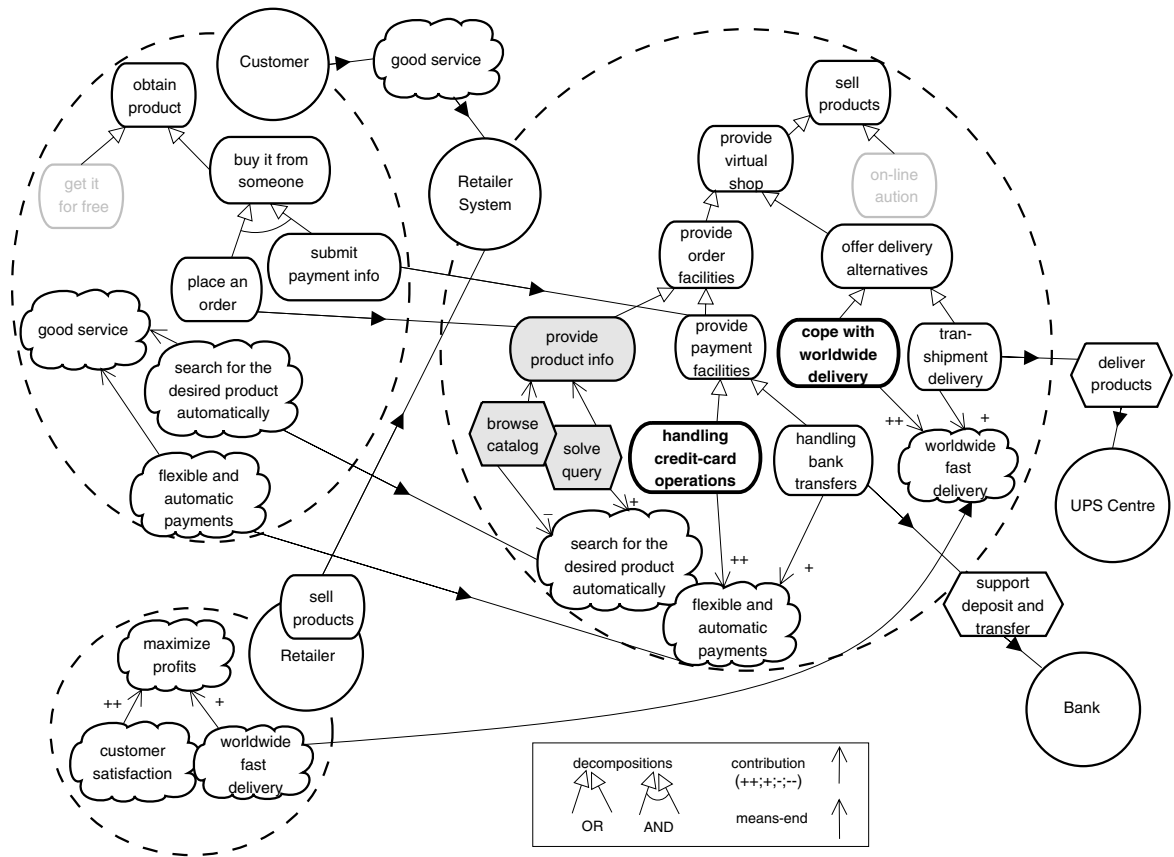


Figure 8. The system-to-be (Retailer System) goals and means-ends analysis according to a top-down approach.

Capability	Means-Ends (goal, plan)	List of Contributions	Type
Cap_1	provide product info, browse catalog	{search for des. product aut. -}	not dependent
Cap_2	provide product info, solve query	{search for des. product aut. +}	not dependent
Cap_3	handling credit-card oper., ??	{flexible and aut. payment ++}	??
Cap_4	hand. bank transfers, sup. deposit and tran.	{flexible and aut. payment +}	dependent on Bank
Cap_5	cope with worldwide del., ??	{worldwide fast del. ++}	??
Cap_6	transshipment, deliver products	{worldwide fast delivery +}	dependent on UPS Centre

Table 1. Retailer System capabilities, at late requirements, after a top-down analysis.

- *dependent on* and *publicized* web services required to the system in order to meet with specific needs but delegated to external actors, e.g. Cap_5 and Cap_6 .

The analyst now has all the domain elements to trace system capabilities back to stakeholders' needs and to measure the coverage of a given need, by different capabilities, as clarified by the following examples.

Example 1 (Simple need). Let us assume the analyst is interested to characterize potential system behaviors (capabilities execution) in respect to specific Customer's needs. Therefore, according to the conceptual models sketched out in Fig. 8, the analyst defines a first need in terms of goals and softgoals, namely, by an *isn* as follows:

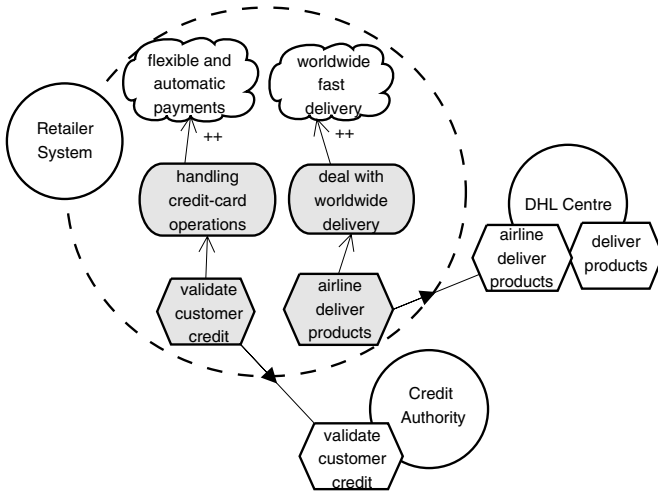


Figure 9. Adding new (external) capabilities (Cap_3, Cap_5) to the Retailer System by a bottom-up approach.

$isn_{1a} = \langle (\text{buy it from someone,}$
 search for the desired product automatically,
 flexible and automatic payments) \rangle

The intuitive meaning of this isn_{1a} corresponds to a class of Customers' needs related to each on-line buy-request along with some preferences modelled by softgoals **search** for the desired product automatically and **flexible and automatic payments**. Notice that also typical quality of service (QoS) specification could have been modelled here in terms of softgoals which connotes stakeholders goals.

The next step for the analyst is to look for the more convenient system capabilities to be executed in order to have a good chance to achieve the isn_{1a} . As illustrated by the conceptual model of Fig. 8, in order to achieve the Customer goal **buy it from someone**, the Customer has to satisfy both the goals **place an order** and **submit payment info**. The conceptual model also allows the designer for an inter-actor analysis, i.e. between Customer and Retailer System. Specifically, by means of goal dependencies, the two previous Customer's goals achievement implies that the system have to satisfy the goals **provide product info** and **provide payment facilities**. Despite of several alternatives cope with such goals— Cap_1 and Cap_2 related to **provide product info** and Cap_3 and Cap_4 related to **provide payment facilities**—the analyst only selects those that maximally contribute to the isn_{1a} softgoals achievement—**search for the desired product automatically** and **flexible and automatic payments**. From

Needs	Capab.	Contribution	ServiceType
isn_{1a}	Cap_2	{search for des. product aut. +}	publicized
	Cap_3	{flexible and aut. payment ++}	not-public.
isn_{1b}	Cap_2	{search for des. product aut. +}	publicized
	Cap_4	{flexible and aut. payment +}	not-public.
...	...	...	...
isn_{2a}	Cap_2	{search for des. product aut. +}	publicized
	Cap_3	{flexible and aut. payment ++}	not-public.
	Cap_5	{worldwide fast delivery ++}	publicized
isn_{2b}	Cap_2	{search for des. product aut. +}	publicized
	Cap_3	{flexible and aut. payment ++}	not-public.
	Cap_5	{worldwide fast delivery +}	publicized
...	...	...	...

Table 2. Querying the Tropos model by (isn).

Tab. 1, to cope with such an isn_{1a} , the more convenient capabilities to execute are Cap_2 and Cap_3 . Moreover, starting from the same needs -i.e. let us consider isn_{1a} and isn_{1b} have the same need definition- we can have different groups of alternatives (capabilities). In other words, in Tab. 2 each isn is characterized by a number and a letter, the first identifies same needs and the second addresses different alternatives in order to cope with the first.

Example 2 (complex need). Let us assume the same Customer's need proposed in the Example 1. But this time, the analyst is also interested to understand when and how the softgoal **worldwide fast delivery** can be satisfied, which reflects the important Retailer's need for maximizing her profit, i.e. modelled by softgoal **maximize profit**. Notice that, this is the case of a complex need, because it derives from two stakeholders. Therefore, let us assume to deal with the following:

$isn_{2a} = \langle (\text{buy it from someone,}$
 search for the desired product automatically,
 flexible and automatic payments),
 (sell products, worldwide fast delivery) \rangle

Proceeding in the same way of Example 1, the inter-model analysis produces the results shown in Tab. 2.

Discussion. As already said, we have focused on the design of alternatives as services the system can provide without dealing with their internal business process relations (e.g. WSDL, BPEL, and SOAP messages), which are, on the contrary, issues more related to the services detailed design and implementation phases. From this perspective, in our approach, each alternative corresponds to a class of services. For example, in our case, Fig.9 shows that Retailer System to satisfy its goal **deal with worldwide delivery**

has to rely on all those service providers —e.g. DHL Centre— that are able to endow with the class of services modelled by airline delivery products.

More precisely, as indicated by the two above examples, the result of such a needs-analysis generates several classes of services that may also require to be composed in order to better accommodate with complex service requests. This is the case of our proposed examples. Indeed, Example 2 shows that **Retailer System** has to provide Cap_2 , Cap_3 , and Cap_5 —i.e. three classes of services— in order to deal with a service request that carries out the need of isn_{2a} . Nevertheless, the composition for such services has to be specified (e.g. the execution order).

Thanks to the proposed capability definition, the ability part allows to specify the elements of a composed service. Moreover, the service execution control-flow can be specified by an AUMML activity diagram, as proposed in [10, 11, 13, 17] for the capability detailed design, so giving a specification of the service orchestration.

6. Related work

Current research in web services design tend to focus on designing service interfaces from existing components by addressing specific analysis and design principles [4, 7, 14, 19]. For instance, [4] describes a method, driven by the cohesion design principle, to derive web services specification from the UML models of the public interfaces for a given class, by combining them in a way to reduce the number of services; [19] addresses the problem of designing web services operations with appropriate level of granularity and differentiates between coarse-grained and fine-grained services.

These general principles are useful as guidelines for the design of service interfaces for a specific application domain. When considering individual services interface design, two design principles are of particular importance: maximizing method (i.e. web services operation) cohesion and minimizing method coupling. [7] proposes a service interface design method based on identifying elementary business functions and on the conversion of standard message (documents) formats into a set of corresponding service interfaces. The approach aims at addressing scalability and flexibility issues.

[14] describes a design methodology for web services and business processes aiming at: (i) determining objectives and describing the business process structure, i.e. the process logical flow; (ii) describing roles, i.e. business activity responsibilities. The methodology recognizes a critical role to business process goal modeling and provides service design guidelines based on the principles of minimizing coupling and maximizing cohesion to ensure that the resulting services are self-contained, modular, extensible and

reusable.

A goal driven approach to service elicitation, distribution and orchestration is proposed in [11]. The basic idea is to understand the needs of different organizations for a new added-value composite service by modeling goals to be achieved by the performance of a process as nodes in a graph, called *map*. The edges of this graph represent *strategies*, that is the ways to achieve the edge ending node. A couple of nodes (goals) and the edge connecting them are mapped to a *system service*. This graph is then partitioned into subgraphs (*distributed maps*), each one corresponding to a specific service provider which is used to support orchestration via the analysis of the dependencies between each pair of sub-maps. This work adopts ideas similar to ours and is moving along parallel directions. However, it addresses different specific problems than those addressed here.

[22] describes a framework for the analysis of automated business interactions among different organizations via web services. In particular, the paper proposes the use of requirements engineering techniques, such as the actor/goal-driven i^* methodology, in conjunction with value-based techniques, such as e^3value framework, in order to model and explore alternative business models for a network of organizations and to evaluate the set of alternatives on their economic viability, leading into the design of the final technical system. During the activities related to the modelling and analysis of the alternatives, a deep use of the goal analysis techniques proposed by the i^* framework are exploited.

7. Conclusion and future work

This paper proposed an extension of the *Tropos* design process which makes it suitable for the design of services. A revised notion of capability allows to explicitly correlate stakeholders needs with system's plans and environmental constraints; novel analysis procedures enhance the *Tropos* design process supporting bottom-up problem solving strategies and capability modelling. The approach has been illustrated on a small case study taken from the literature. The purpose of this illustration was to show how service design can be conceived in terms of modelling system capability *alternatives*, which can be directly *traced* back to consumer needs and classified according to needs-coverage metrics. This offers a roadmap, at design time, that relates user goals to collections of services available in different directories.

Future research will focus on further refinements of the service design process and on the environment that supports it. In particular, we intend to study how design time knowledge on how the alternative services meet stakeholder needs can be exploited by monitoring and compensation mechanisms at run time, for instance, by giving a way to anticipate

and manage possible failure conditions through the selection of a more convenient alternative among the available services.

References

- [1] Web service definition language (wsdl), <http://www.w3.org/tr/wsdl>.
- [2] M. Aiello, M. Aoyama, F. Curbera, and M. P. Papazoglou, editors. *Service-Oriented Computing - ICSOC 2004, Second International Conference, New York, NY, USA, November 15-19, 2004, Proceedings*. ACM, 2004.
- [3] M. Aiello and P. Giorgini. Applying the Tropos Methodology for Analysing Web Services Requirements and Reasoning about Qualities of Services. *Journal for the Informatics Professional*, 5(4), 2004.
- [4] S. W. Ambler. Deriving Web services from UML models, Part 1. <http://www-106.ibm.com/developerworks/webservices/library/ws-uml1/>.
- [5] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, July 2004.
- [6] J. Castro, M. Kolp, and J. Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems*. Elsevier, Amsterdam, the Netherlands.
- [7] G. Feuerlicht and S. Meesathit. Design framework for interoperable service interfaces. In Aiello et al. [2], pages 299–307.
- [8] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. Model checking early requirements specifications in Tropos. In *IEEE Int. Symposium on Requirements Engineering*, pages 174–181, Toronto (CA), Aug. 2001. IEEE Computer Society.
- [9] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Formal Reasoning Techniques for Goal Models. *Journal on Data Semantics*, 2004.
- [10] IBM. Business process execution language for web services. <http://www.ibm.com/developerworks/library/ws-bpel>.
- [11] R. S. Kaabi, C. Souveyet, and C. Rolland. Eliciting service composition in a goal driven manner. In Aiello et al. [2], pages 308–315.
- [12] R. Kazhamiakin, M. Pistore, and M. Roveri. A Framework for Integrating Business Processes and Business Requirements. In *Proc. of EDOC 2004*, Monterey, CA, USA, September 2004.
- [13] D. Lau. Designing Web Services with Tropos. Master’s thesis, Dept. of Computer Science, University of Toronto, Canada, 2005.
- [14] M. P. Papazoglou and J. Yang. Design Methodology for Web Services and Business Processes. In *TES ’02: Proc. of the Third International Workshop on Technologies for E-Services*, volume 2444 of LNCS, pages 54–64, London, UK, 2002. Springer-Verlag.
- [15] L. Penserini and J. Mylopoulos. Design Matters for Semantic Web Services. Technical Report T05-04-03, ITC-irst, Trento, Italy, <http://sra.itc.it/tools/taom4e/relwork>, April 2005.
- [16] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. From Capability Specifications to Code for Multi-Agent Software. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2006.
- [17] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. From Stakeholder Intentions to Software Agent Implementations. In *Proceedings of the 18th Conference On Advanced Information Systems Engineering (CAiSE’06)*, number 4001 in LNCS. Springer-Verlag, 2006.
- [18] A. Perini, A. Susi, and J. Mylopoulos. Agent-Oriented Design for Web Services. Technical report, ITC-irst, Trento, Italy, <http://sra.itc.it/tools/taom4e/relwork>, 2005.
- [19] M. Stevens. Multi-Grained Services. <http://www.developer.com/design/article.php/1142661>.
- [20] A. Susi, A. Perini, P. Giorgini, and J. Mylopoulos. The Tropos Metamodel and its Use. *Informatica*, 29(4):401–408, 2005.
- [21] P. Traverso, M. Pistore, M. Roveri, A. Marconi, R. Kazhamiakin, P. L. P. Busetta, and P. Bertoli. Supporting the Negotiation between Global and Local Business Requirements in Service oriented Development. Technical Report T04-08-03, ITC-irst, Trento, Italy, <http://sra.itc.it/tools/taom4e/relwork>, 2004.
- [22] B. van der Raadt, J. Gordijn, and E. Yu. Exploring Web Services from a Business Value Perspective. In *Proceedings of the 13th International Requirement Engineering Conference (RE’05)*, pages 53–62. IEEE Computer Society, 2005.
- [23] C. Walton. Uniting agents and web-services. In *AgentLink News*, number 18. 2005.
- [24] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, University of Toronto, 1995.