# Designing Web Services with Tropos

Diana Lau and John Mylopoulos
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada M5S 3G6
{dianalau, jm}@cs.toronto.edu

## Abstract

*We propose a methodology for designing web services. The methodology is founded on Tropos [Perini01, Castro02], an agent-oriented software development technique, and supports early and late requirements analysis, as well as architectural and detailed design. An online retailer example is used for illustration of the proposed methodology. We also compare the generated design with a sample design presented in [BPT01].*

## 1. Introduction

*Web services* are self-contained, self-described, modular applications that can be published, located and invoked across the Internet. Web services are being touted as the next revolution of the World Wide Web. Not surprisingly, many organizations are jumping on the web service bandwagon, committing people and resources to their development. To a large extend, the focus of this development has been the functionalities of web services and their integration with existing software systems. Unfortunately, such a focus only answers "what" and "how" questions, but ignores the "why" questions: Why is this service being offered? Why is it designed the way it is? Whom is it intended to serve? Is it offering the service in a way that is acceptable to all members of its intended user group, or just some? The objective of this paper is to fill in this gap by proposing a design methodology for web services, adopted from the Tropos project.

Tropos [Perini01, Castro02] is an agent-oriented software engineering methodology that spans early and late requirements, as well as architectural and detailed design. Tropos is founded on the concepts of *actor*, *goal* and (actor) *social dependency* in the same sense that UML is founded on the notions of *object*, *class*, *method*, *inheritance* and the like. Until now, the focus of the Tropos project has been the design of agent-oriented, distributed, open, applications. Applications in distributed systems are similar to web services. However, once a web service is registered on the web, the service provider has no control on how and where the service would be used; therefore, more consideration has to be taken into account, such as interoperability, accessibility and customizability. The purpose of this paper is to adopt the Tropos methodology to the design of web services. An online retailer business scenario is used as a case study for illustration purposes.

The rest of the paper is organized as follows. Section 2 introduces terminology to be used extensively throughout this paper, and describes the online retailer case study. In Section 3, we illustrate how the methodology is applied for the requirements and architectural design phases, while Section 4 continues the case study with later stages of the development cycle. The last sections outline future work on a design methodology for web services and summarize the contributions of this research.

## 2. Preliminaries

Throughout this paper, we use terminology adopted from Tropos. In particular:

***Actor.*** An actor models a stakeholder for the new system, or a component of the system itself. An actor has strategic goals and intentions and can carry out actions to fulfill them. An actor can be a physical, social or software *agent*, a *role* or a *position*. An agent can occupy a position, while a position covers several roles [Perini01].

***Goal.*** A goal represents an actor's strategic interests. There are *hard* and *soft* goals. Soft goals do not have a clear-cut definition and/or criterion for deciding whether they are satisfied or not, and often model non-functional requirements.

***Social Dependency.*** A social dependency is a relationship between two actors where one actor, the *depender*, depends on another actor, the *dependee*, to deliver a *dependum*, by achieving a goal, executing a plan, or delivering a resource [Perini01].

***Capability.*** Capability represents the ability of an actor to define, choose and execute a plan for the fulfillment of a goal.

We use an online retail store from the Business Process Team [BPT01] to illustrate how to apply the Tropos methodology. Our aim is to apply our proposal to the original example, and compare our results with the worksheet-based methodology used in [BPT01].

The online retail store sells a range of products. Customers can buy goods using a computer, a cell phone, or a Personal Digital Assistant (PDA) through the Internet. After an order is placed, the retailer contacts the Credit Authority to validate customer credits. If the credit is valid, the order will be confirmed, and the retailer will then charge the customer through the Credit Authority to the customer's bank account.

Once payment is processed, the retailer notifies the Direct Supply Vendor (DSVendor) in order to provide order and delivery information. The DSVendor collects goods from its inventory, and ships them to the transport center together with the delivery information.

Eventually, the transport center delivers the ordered products to the customer. Upon completion of the delivery, the center informs the retailer.

An example of a Tropos model is shown in Figure 1. *Customer* has a goal to own products and a soft goal to get them at the lowest cost. On the other hand, *Retailer* has a soft goal to maximize profit and depends on *Customer* for the soft goal of having repeatable business.

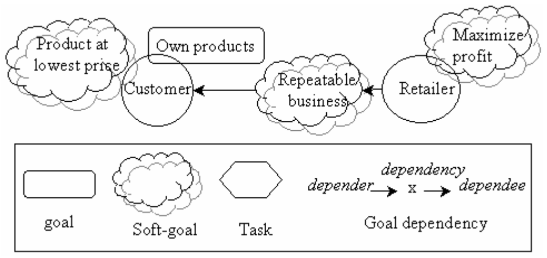

Figure 1. Goal dependencies between Customer and Retailer

Figure 2 shows how the "own products" goal can be met through co-operation with *Retailer*. This goal can be achieved by "get for free", "buy from someone" or "exchange with other goods". The bottom of this goal hierarchy lists the tasks that are necessary to satisfy goals located higher up in the hierarchy. To demonstrate this, Figure 3 exhibits that different payment methods can be used, whereas Figure 4 illustrates alternatives for submitting customer information.

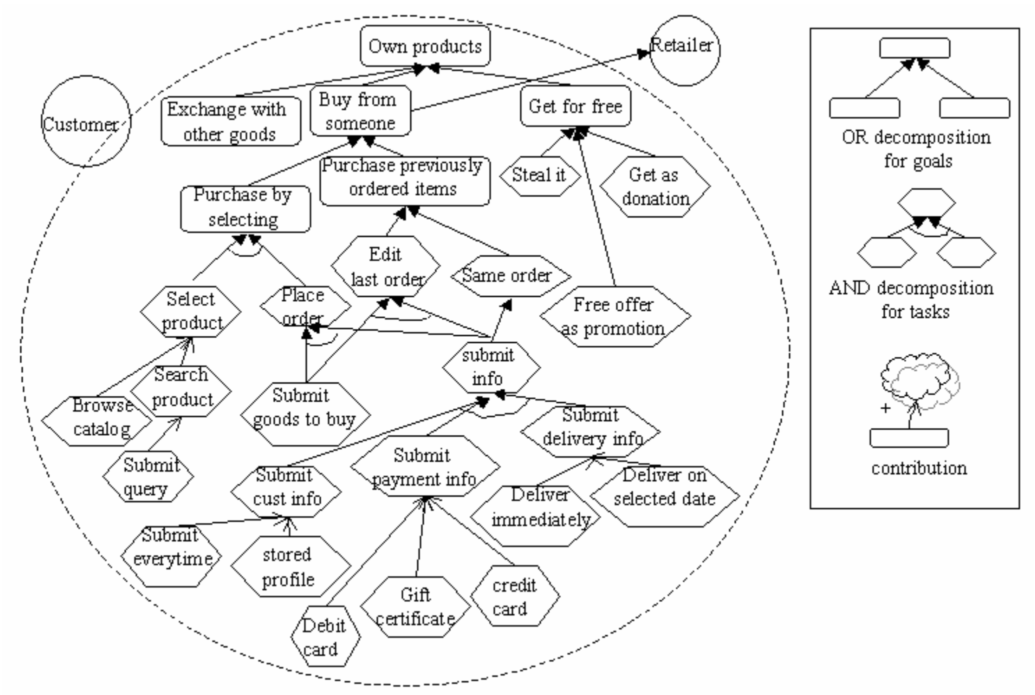

Figure 2. Strategic Rationale Diagram for Customer to own products

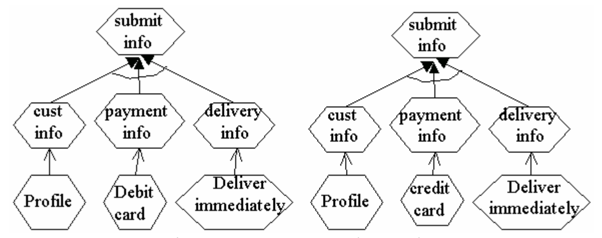

Figure 3. Payment alternatives

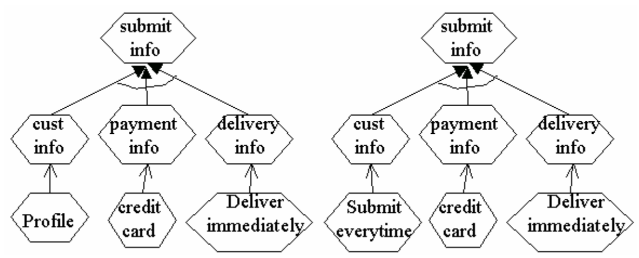

Figure 4. Alternatives of submitting customer information

## 3. Early Requirements Analysis

During the requirements phase, stakeholders and goals for the existing organizational setting are first identified, and then the functional and non-functional requirements of the system-to-be are defined. Subsequently, the design stages focus on the system specification, according to the requirements gathered from earlier stages [Castro02].

During early requirements analysis, the intentions of the stakeholders are identified and analyzed [Perini01]. During this phase, the system-to-be is not represented or discussed, yet. In particular, the phase includes steps as follows.

**Step 1: Identify stakeholders**

In this example, there are six stakeholders as follows:

- *Customer*, shops online through retailer system;
- *Retailer*, sells products;
- *Direct supply vendor*, supplies goods to *Retailer*;
- *Transport Center*, delivers goods to *Customer;*
- *Credit Authority*, who validates *Customer*'s credits and charge them from the bank;
- *Bank*, supports withdrawal, deposit and transfer of money between *Customer* and *Retailer*.

**Step 2: Identify goals, other actors, and dependencies**

Top-level stakeholder goals are identified, analyzed and decomposed. When an actor needs another actor to achieve a goal, a social dependency is established between them. The output of this process is an actor diagram.

Figure 5 is an actor diagram depicting stakeholders and their interests. Specifically, *Customer* has a goal to own products and soft goal to obtain products at the lowest price. He depends on *Retailer* to receive good customer service. Quality of customer service is a soft goal. Conversely, *Retailer* has a goal dependency on *Customer* for repeatable business, also wants to maximize profit. *Direct Supply Vendor* has a soft goal to maximize profit, depends on *Retailer* to offer products, but also depends on *Transport Center* to ship goods to C*ustomer*. *Transport Center* is associated with a task (deliver goods) and a soft goal (maximize profit). "Deliver goods" is a task in the sense that there is a standard way of achieving it. On the other hand, *Credit Authority* and *Bank* are less relevant actors. The former is associated with "validate customers' ability to pay", whereas the latter is associated with two soft goals, which are secure transaction and maximize profit, and the task "support basic banking transaction".

**Step 3: Conduct means-end analysis**

During this step, goals are further decomposed and positive/negative contributions among them are specified [Perini01]. Tasks can also be decomposed into simpler tasks as well. In the case study, we focus on

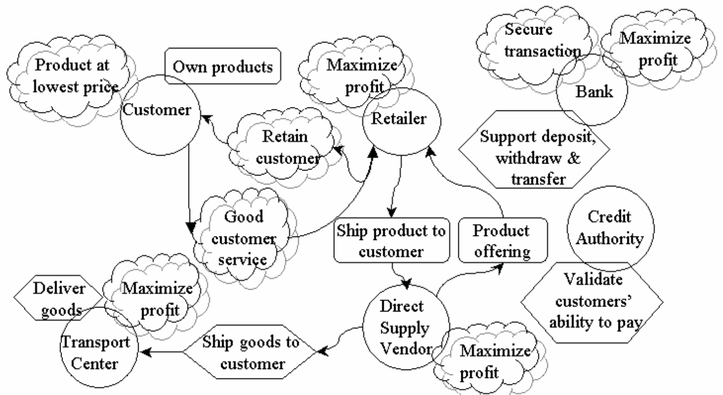

Figure 5. Actor diagram modeling the stakeholders of online retailer business model

*Customer*, *Retailer System* and *Direct Supply Vendor*. The output of this step is a goal (or, strategic rationale) diagram for each stakeholder. For example, Figure 6 shows the strategic rationale diagram for "maximize profit".

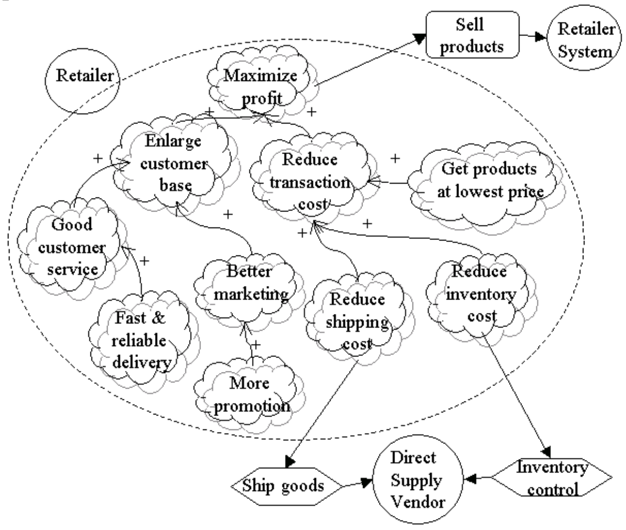

Figure 6. Strategic Rationale model for "maximize profit"

## 4. Late Requirements Analysis

The system-to-be is introduced in this phase and treated as a new actor with delegated hard and soft goals. In the case study, *Retailer System* is introduced to fulfill the "sell product" goal for *Retailer*.

Figure 7 shows *Retailer System* and its responsibilities. In particular, *Retailer* delegates "sell product" to *Retailer System.* This goal can be achieved in several ways, e.g., through self-service (amazon.com), quotation (bmw.com), auction (eBay.com), and salesperson (bmo.com for bank plan selector). These sub-goals are then further decomposed into finer goals/tasks. For example, "self serve" is decomposed into "cataloguing" -- includes listing and searching products -- and "handle order" -- includes dealing with deliveries and payments. On the other hand, "charge customer" is delegated to *Credit Authority*, since it needs credit validation for *Customer.*

## 5. Architectural Design

This phase defines the system's global architecture in terms of subsystems (actors) interconnected through data and control flows (dependencies) [Perini01].

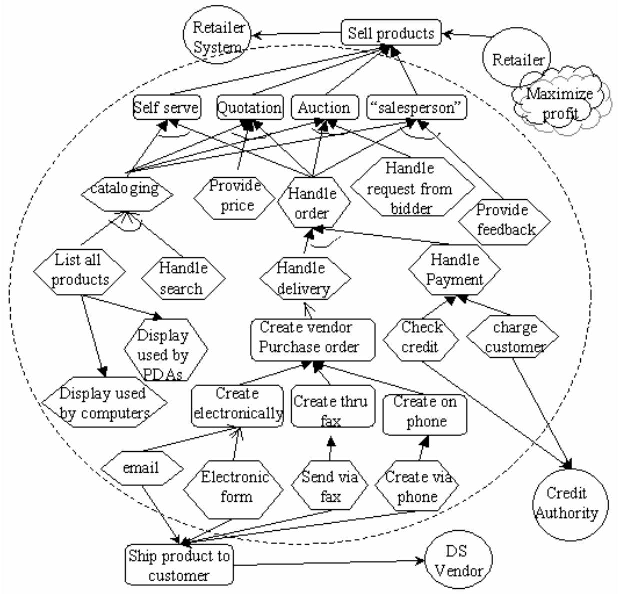

Figure 7. Goal diagram for the Retailer System actor

**Step 1: Create extended actor diagram**

The actors introduced in this step are all system actors, representing subsystems or components of the system-to-be. The resulting architecture is shown in Figure 8.

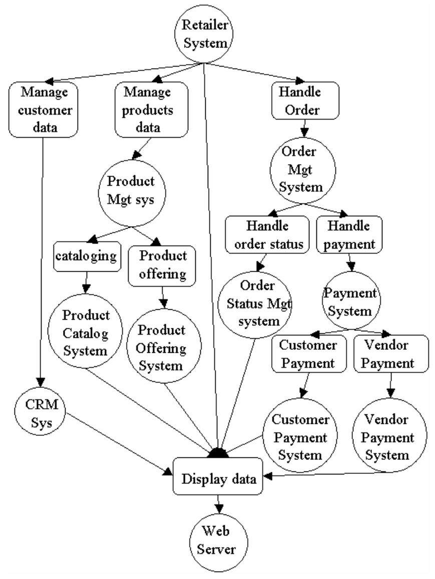

Figure 8. Actor diagram for the new online retailer system architecture

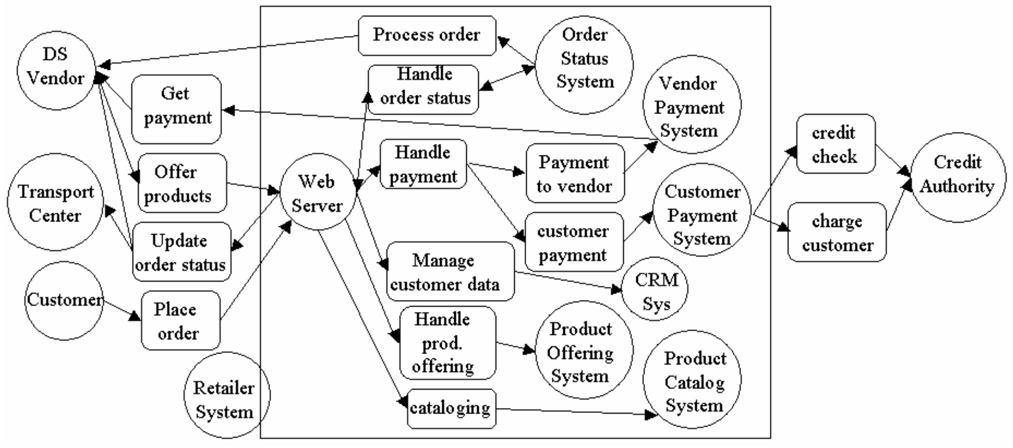

Figure 9. Extended actor diagram with respect to the Retailer System

The architecture consists of four main subsystems: *Customer Relationship Management System, Product Management System, Web Server,* and *Order Status System.* These are responsible for handling customer information, products, online shopping and orders, respectively. Within *Product Management System*, there are *Product Catalog* and *Product Offering* systems, which supports inventory control and product offerings.

Consequently, an extended actor diagram is produced to demonstrate how each subsystem fits in a big picture. According to Figure 9, *Customer*, *Transport Center*, *Direct Supply Vendor* and *Credit Authority* are external users of the retailer system.

**Step 2: Identify actors' capabilities**

This step identifies the capabilities needed by system actors in order to fulfill their assigned goals and plans [Castro02]. Capabilities are identified by analyzing actor models obtained in previous steps. Each dependency relationship requires one or more capabilities triggered by external events. In the retailer system goal diagram, we have modeled all possible ways of satisfying top-level goals. From these we generate a list of capabilities that are necessary in order to achieve the top-level goal.

Table 1 shows the capabilities of some subsystems. The middle column "N" is used to name the capabilities in numbers. These numbers will be used in the next step when assigning capabilities to agents.

**Step 3 Assign capabilities to agents**

After identifying actor capabilities, a set of agent types is defined, and each of them is assigned one or more capabilities [Castro02]. Table 2 shows the agent types and their capabilities. The agent *Web Server* possesses the capabilities required for *Customer Interface System* and *External Interface System*, as defined in the previous steps.

Table 1. Actors' capabilities

| Actors | N | Capabilities |
|---|---|---|
| Customer Relationship Management | 1 | get customer profile |
| | 2 | create customer profile |
| | 3 | update customer profile |
| | 4 | record customer order |
| | 5 | remove customer order |
| | 6 | retrieve order history |
| Product Offering System | 13 | accept/refuse product offering via email |
| | 14 | accept/refuse product offering via electronic form |
| | 15 | receive product offering notifications via email |
| | 16 | receive product offering notifications via electronic form |
| | 17 | receive product offering notifications via fax |

Table 2. Agent types and their capabilities

| Agent | Capabilities |
|---|---|
| Inventory Control System | 1, 2, 3, 4, 5, 6 |
| Product Management System | 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23 |

## 6. Later Stages

One can now derive the Retailer System architecture, agent types and their capability list from the output of the phases outlines in previous sections. To continue with the rest of the development cycle, it is suggested that the Agent-based Unified Modeling Language (AUML) notation be used when adding details to all architectural components in the detailed design phase. The Capabilities Diagram models capabilities from the point of view of a specific agent. Since such diagrams model each event as a transition, a UML activity diagram is suitable in order to describe such events. On the other hand, UML sequence diagrams are recommended for modeling communications between agents. Since our design is the same as the one mentioned in [BPT01], it is not necessary to repeat the details in this paper.

At the end of the design process, open standards can be used for the implementation of the web services. These are the Web Services Definition Language (WSDL) [WSDL] for describing a web service, Simple Object Access Protocol (SOAP) [SOAP] for representing remote procedure calls/response, and Universal Description, Discovery Integration (UDDI) [UDDI] for describing and discovering web services.

By using the capabilities assigned to each agent, one can identify the data types and activities that are involved in achieving system goal. Since we are interested in designing web services, XML Schema (XSD) and WSDL are used to describe relevant data types and activities. Figure 10 shows the Customer complex type defined in terms of an XML Schema.

```
<complexType name="Customer">
    <all>
      <element name="address" nillable="true"
type="string"/>
      <element name="lastName" nillable="true"
type="string"/>
      <element name="email" nillable="true"
type="string"/>
      <element name="firstName" nillable="true"
type="string"/>
      <element name="customerID"
nillable="true" type="string"/>
    </all>
</complexType>
```

Figure 10. XML Schema for Customer

To give more details, we focus on the *Customer Relation Management* subsystem. It is responsible for creating, deleting and updating customer profile, as well as updating order history. Each capability is considered as a WSDL operation embedded in a port type for an agent. This is demonstrated in Figure 11.

```
<message name="getCustomerProfileRequest"/>
<message name="getCustomerProfileResponse">
    <part name="result" type="xsd1:Customer"/>
</message>
<message name="addOrderToOrderHistoryRequest">
    <part name="order" type="xsd2:Order"/>
</message>
<message name="addOrderToOrderHistoryResponse">
    <part name="result" type="xsd:boolean"/>
</message>
...
<portType name="CRM">
    <operation name="getCustomerProfile">
      <input
message="tns:getCustomerProfileRequest"
name="getCustomerProfileRequest"/>
      <output
message="tns:getCustomerProfileResponse"
name="getCustomerProfileResponse"/>
    </operation>
    <operation name="addOrderToOrderHistory"
parameterOrder="order">
      <input
message="tns:addOrderToOrderHistoryRequest"
name="addOrderToOrderHistoryRequest"/>
      <output
message="tns:addOrderToOrderHistoryResponse"
name="addOrderToOrderHistoryResponse"/>
    </operation>
...
</portType>
```

Figure 11. Web service definition for Customer Relationship Management system

## 7. Discussion

Combinations of leaf goal nodes in an actor diagram constitute alternatives for achieving root goals. We are going to use the eight use cases [BPT01] to compare the solutions generated from our methodology and the worksheet-based methodology presented in [BPT01].

In the “firm sales order” business process use case, the worksheet-based method requires customers to complete all personal identity data, select products to purchase, and then accept the terms of sales. On the other hand, our technique supports several alternatives for providing customer and order information. According to Figure 13, customer information can be obtained from a customer submitting this information with every purchase, or by maintaining a customer profile for repeat customers. Moreover, products to be purchased can be specified in three ways: select products without knowledge of previous purchases, purchase same products as in the last order, or edit the last order. Taking all these combinations into account, our design offers six alternatives for use case.

In the “customer credit payment” use case, *Credit Authority* makes a credit charge against the customer’s account, and then reports to *Retailer* about the status of the charge. It is assumed that credit card is the only way to pay the bill. Nonetheless, our analysis in Figure 13 shows that debit cards and gift certificates are alternative forms of payment.

The result for the “customer credit inquiry” use case is the same for both methods, since this task is delegated to *Credit Authority* in our design. Both methods generate the same results for the “purchase order” and “inventory management” use cases as well. The “present invoice” is the same in both techniques too, because this “update delivery status” task is delegated to *DSVendor* as shown in Figure 12. The “ship goods” use case is not comparable since our retailer system does not involve communication between the *DSVendor* and *Transport Center*. Finally, in the “sales product notification” use case, a product specification request is sent to *Retailer*. The description in [BPT01] does not specify how the request is sent; however, it can be sent via email or some electronic form in our method.

Comparing our design with the one shown in [BPT01], we find that our design is comparable but more generic, and therefore more customizable. Alternatives for achieving each goal are determined and analyzed during means-end analysis. As a result, web services designed with the Tropos methodology take into account all possibilities for satisfying root goals of major stakeholders. On the other hand, the processes described in [BPT01] only consider one solution for satisfying root-level goals.

Apart from the use cases mentioned above, soft goals, such as maximize profits and buy products at the lowest price, are not considered in the worksheet-based technique. Our design includes an analysis of these goals and even converts some into hard goals and tasks that are executable by *Retailer System*. This is a bonus offered by our proposed methodology. Taking the *Customer*’s soft goal of buying products at the lowest price as an example, it can be achieved by two hard goals: promotion and proof availability, as illustrated in Figure 13. The system-to-be can then be designed to show the competitor’s price in the product catalog and offer discount for large quantity purchase. It can also remind the *Retailer* to select products on sale periodically.

Besides, the Tropos methodology satisfies two fundamental design principles for services: coupling and cohesion [Papaz02]. During the requirements phase, top-level goals are identified, analyzed and then refined into subgoals and tasks. Each goal and task is evaluated independently. Therefore, each group of activities is loosely-coupled with respect to other activity groups. In terms of cohesion, events in each web service produced by Tropos are functionally cohesive, since they all contribute to a specific goal or task. The design in [BPT01] satisfies these criteria as well. However, it only accommodates one way of completing each required task, and alternative ways are ignored.

## 8. Conclusions

The paper has proposed Tropos as a design methodology for web services. The proposal was illustrated with an online retailer case study adopted from the literature. The key idea of the methodology is that software services are designed by starting from stakeholder goals, and by analyzing these goals in order to define the space of alternative solutions. The web service design generated from this process, is expected to accommodate as many of those solutions as possible, thereby rendering the design more generic and usable by a broader class of applications.

For future work, we plan to extend Tropos so that it is tailored specifically to web service design. After all, Tropos does not support mechanisms for making software platform-, language-, and implementation-independent. We also propose to develop mechanisms that lead to software designs that accommodate a variety of users, including ones who lack skills, e.g., blind users, or users with motor control problems. In addition, we propose to study design techniques that address problems of unreliable communication and unpredictable loads [Bosw01].

A key concept in the design of web services is that of a business process [Papaz02]. Hence, we’d like to explore the integration of Tropos with the open standard business process language, Business Process Execution Language (BPEL), to describe how tasks work together to achieve a goal, instead of using Agent Interaction and Capability diagrams.

## 9. References

[BPT01] Business Process Team, Business Process Analysis Worksheets & Guidelines v1.0. Appendix C.

[Bosw01] Bosworth, A., “Developing Web Services”, *Data Engineering, 2001. Proceedings. 17th International Conference on, 2-6 April 2001.*

[Castro02] Castro, J., Kolp, M., Mylopoulos, J., “Towards requirements-driven information systems engineering: the Tropos project”, *Information Systems Journal, 2002.*

[Kirda01] Kirda, E.; Jazayeri, M.; Kerer, C.; Schranz, M.; “Experiences in Engineering Flexible Web Services”, *Multimedia, IEEE, Volume: 8 Issue:1, Jan-March 2001*

[Papaz02] Mike P. Papazoglou, Jian Yang, “Design Methodology for Web Services and Business Processes”, *Proceedings of the 3rd VLDB-TES Workshop, August, Hong Kong, Lecture*

*Notes in Computer Science Vol. 2444, Springer, 2002.*

[Perini01] A. Perini, P. Bresciani, F. Giunchiglia, P. Giorgini, J. Mylopoulos, "A Knowledge Level Software Engineering Methodology for Agent Oriented Programming", *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, May 2001.

[WSDL] "Web Service Definition Language (WSDL) 1.1", http://www.w3.org/TR/WSDL

[SOAP] "Simple Object Access Protocol (SOAP) 1.1", http://www.w3.org/TR/SOAP

[UDDI] "Universal Description, Discovery, Integration", http://www.uddi.org

## 10. Appendix

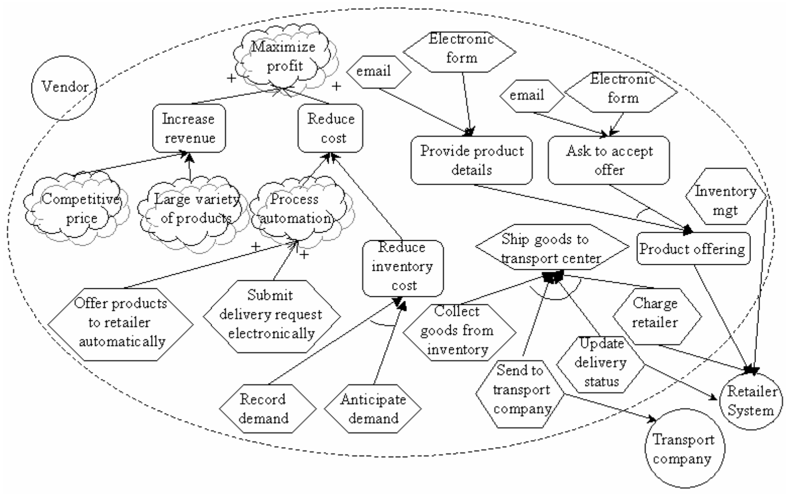

Figure 12. Goal diagram for Vendor

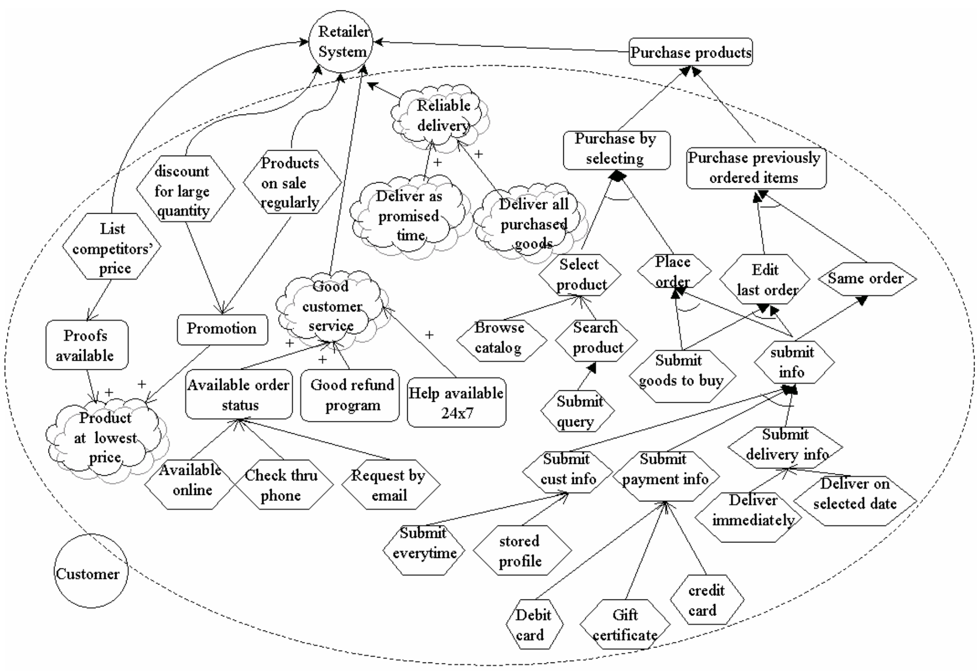

Figure 13. Goal diagram for Customer