APPLYING AGENT-ORIENTED MODELLING AND PROTOTYPING TO SERVICE-ORIENTED SYSTEMS

Aneesh Krishna, Ying Guan, Aditya Ghose
Decision Systems Laboratory, School of Information Technology and Computer Science
University of Wollongong, NSW 2522, Australia

Keywords: Service-Oriented Architecture (SOA), Web Engineering, Agent-Oriented Conceptual Modelling, Executable Specification.

Abstract: A Service-Oriented Architecture (SOA) is a form of distributed systems architecture, which is essentially a collection of services. Web services are built in the distributed environment of the Internet, enabling the integration of applications in a web environment. In this paper, we show how agent-oriented conceptual modelling techniques can be used to model service-oriented systems and architectures and how these models can be executed. The resulting executable specification environment permits us to support early rapid prototyping of the service-oriented systems, at varying levels of abstraction.

1 INTRODUCTION

A Service-Oriented Architecture (SOA) is an information technology approach or strategy in which applications make use of (or perhaps more accurately, rely on) services available on the network such as the World Wide Web. A SOA is a form of distributed systems architecture, essentially a collection of services. SOA provides consistent interoperability and reuses existing services where possible. Implementing a SOA can involve developing applications that use services and making applications available as services. A SOA is typically characterized by the following properties (Jørstad 2005): Logic view, Message orientation, Description orientation, Granularity and Platform neutrality. Web services are self-describing software applications that can be advertised, located, and used across the Internet using a set of standards such as SOAP, WSDL (Chinnici 2002) and UDDI (UDDI 2002). Web services are built on the distributed environment of the Internet.

Agents are components that aim to convey models inspired from real life (Chen 2004). The development of agent-based systems offers a new and existing paradigm for the production of sophisticated programs in a dynamic and open environment, particularly in distributed domains such as a web-based systems and electronic commerce (Lohse 1998). Current works on Web services are intimately entwined with work on agent-based systems. Agent-oriented techniques show a potential for web services, where agents are needed both to provide services and to make best use of the resources available (Chen 2004). Early-phase Requirement Engineering (RE) activities of web services are usually performed informally and without much tool support. The agent-oriented conceptual modelling notation as exemplified by the i* framework (Yu 1995) is a popular means of modelling proposed system requirements. Each component of a web service can be regarded as an agent and the whole web service can be viewed as composed of agent system. Hence, we feel that agent-oriented conceptual modelling technique i* framework is suitable for modelling a web service. The i* framework allows analysts to construct agent-based prototypes of the proposed web services based on the preliminary requirements from the stakeholders. Such notations model organizational context and offer high level of social/anthropomorphic abstractions (such as goals, tasks, softgoals (Chung 2000) and dependencies) as modelling constructs. It has been argued that such notations help to answer questions such as; what goals exist, how key actors depend on each other and what alternatives must be considered.

The objective of this paper is to show how agent-oriented conceptual modelling techniques (i* framework) can be used to model service-oriented systems and architectures, and how these models can be executed by mapping i* models into 3APL

Krishna A., Guan Y. and Ghose A. (2006).
APPLYING AGENT-ORIENTED MODELLING AND PROTOTYPING TO SERVICE-ORIENTED SYSTEMS.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - ISAS*, pages 246-252
DOI: 10.5220/0002469902460252

(Hindriks 1999) agents. This approach makes use of the advantages of *i** for the early-phase of requirement engineering and validates the model by mapping it into an executable specification to see the design result in an emulation program.

The remainder of this paper is organized in the following manner. In Section 2, steps for modelling agent-based prototyping of Service-oriented Architecture are given. We shall provide the executable specifications for the i* framework in section 3. Section 4 provides an example to illustrate the proposed approach and section 5 presents some concluding remarks.

2 AGENT-BASED PROTOTYPING OF SERVICE-ORIENTED ARCHITECTURE

2.1 Preliminaries

Many modelling techniques tend to address “late-phase” requirements while the vast majority of critical modelling are arguably taken in early-phase requirements engineering (Yu 1995). Agent-oriented conceptual modelling offers an interesting approach in modelling the early-phase requirements. The i* modelling framework is a semi-formal notation built on agent-oriented conceptual modelling. The central concept in i* is the intentional actor agent. Intentional properties of an agent such as goals, beliefs, abilities and commitments are used in modelling requirements. The actor or agent construct is used to identify the intentional characteristics represented as dependencies involving goals to be achieved, tasks to be performed, resources to be furnished or softgoals (optimization objectives or preferences) to be satisfied. The i* framework also supports the modelling of rationale by representing key internal intentional characteristics of actors/agents. The i* framework consists of two modelling components: Strategic Dependency (SD) Models and Strategic Rationale (SR) Models.

The SD model consists of a set of *nodes* and *links*. Each node represents an “*actor*”, and each link between the two actors indicates that one actor depends on the other for something in order that the former may attain some goal. An SR model represents the internal intentional characteristics of each actor/agent via task decomposition links and means-end links. The task decomposition links provide details on the tasks and the (hierarchically decomposed) sub-tasks to be performed by each actor/agent while the means-end links relate goals to the resources or tasks required to achieve them. The SR model also provides constructs to model alternate ways to accomplish goals by asking why, how and how else questions. Readers are encouraged to read (Yu 1995) for details on i* framework.

2.2 Early Requirements Analysis

During the requirements elicitation phase, stakeholders and goals for individual service are first identified, then the functional and non-functional requirements of each of them are defined and finally the relationships between them are identified. In (Lau 2004), the authors have proposed an approach based on the Tropos methodology (Castro 2002), for designing web services. Our proposal is different from theirs in the sense that, we focus on modelling service-oriented systems and architectures in the early requirement phase, and validate these models by executable specifications, while in (Lau 2004), the authors have proposed the methodology for the whole requirement phase and they aim on implementing the web services.

We shall use the example of online shopping service throughout the rest of this paper to illustrate how to model a web service using i* framework and consequently how these models can be executed.

The online shopping service sells a range of products. Customers can buy goods through a website. After an order is placed, the retailer contacts the payment system to validate customer credits and also charge the customer from the customer’s account. Once payment is processed, the retailer notifies the product management system to provide the necessary information. The product management system collects goods and ships them to the transport centre together with the delivery information. Eventually, the transport centre delivers the ordered products to the customer. Upon completion of the delivery, the retailer will get the confirmation of delivery.

Step 1: Identify actors

Five actors are identified during this step. Customer/Web server, shops online through the website. Retail system, provides service for selling the products. Product management system offers goods and handles delivery. Transport system, delivers goods to the Customer. Payment system validates the Customer’s credits and charges their account.

Step 2: Identify goals

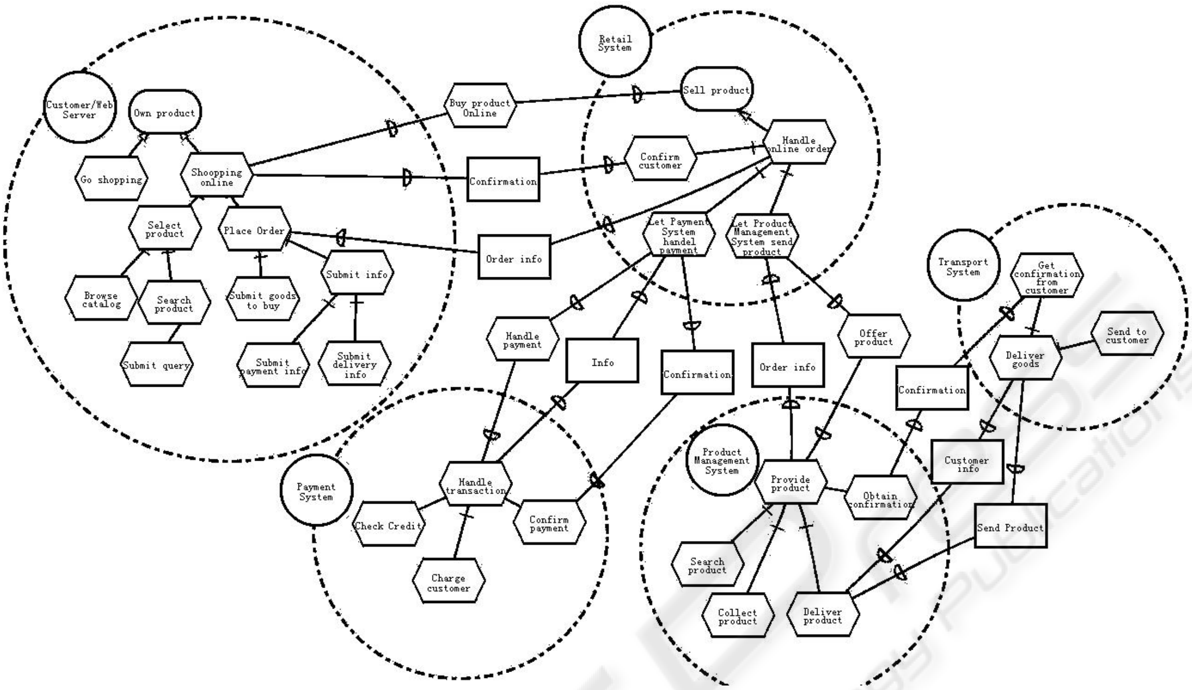

Figure 1: Strategic Rationale Model of online shopping service.

After identifying the actors of a web service, their goals are defined simultaneously. For e.g., the actor Customer/Web server has the goal own product online and actor Retailer system's goal is to sell product.

Step 3: Identify dependency relationships

The actor or agent construct is used to identify the intentional characteristics represented as dependencies involving goals to be achieved, tasks to be performed, resources to be furnished or softgoals (optimisation objectives or preferences) to be satisfied (Lau 2004). Combining the results from steps 1 and 2, the output of this process is a Strategic Dependency (SD) model. Specifically, The customer has a goal to own products, shopping confirmation and softgoal to obtain products at the lowest price and assure the security of credit. He depends on the retail system to receive shopping confirmation. Conversely, the retail system depends on the web server to provide the order information for further transactions. The retail system also depends on the payment system and the product management system to fulfil charging customer Task dependency and the resource dependency providing products to customer respectively. Simultaneously, the payment system needs the customer credit information to charge customer. The product management system depends on the retail system to offer order information, which includes product information, and delivery information, and also depends on the Transport system to ship goods to customer on the condition that delivery information is provided together with goods (to be transported) to the transport system.

The SD model provides an important level of abstraction for describing systems in relation to their environments, in terms of intentional relationships among them. This allows the analyst to understand and analyse new or existing organizational and system configurations even if the internal goals and beliefs of individual agents are not known.

Step 4: Conduct means-end analysis and task-decomposition analysis

In the i* framework, the Strategic Rationale (SR) model provides a more detailed level of modelling by looking "inside" actors to model internal intentional relationships. Intentional elements (goals, tasks, resources, and softgoals) appear in the SR model not only as external dependencies, but also as internal elements linked by task-decomposition and means-ends relationships (Figure 1). Task decomposition links provide details on the tasks and the (hierarchically decomposed) sub-tasks to be performed by each actor while the means-end links relate goals to the resources or tasks required to achieve them. The SR model also provides constructs to model alternate ways to accomplish goals by asking why, how and how else questions. During this step, goals are further decomposed. Tasks can also be decomposed into subtasks. The output of this step is a SR diagram for each actor. Figure 1 shows the SR model for this case. For example, the Customer/Web Server actor, has an internal task to perform ShoppingOnline. This task can be performed by subtasks SelectProduc*t* and PlaceOrder (these are related to the parent task via task decomposition links). The SelectProduct task is further decomposed into subtasks BrowseCatalog and SearchProduct.

After performing outlined four steps, models of the proposed web service are generated. Our next step is to show how these agent-oriented models can be executed. In our proposal, we use 3APL as the programming language for generating executable specifications.

3 EXECUTABLE SPECIFICATIONS FOR I *

3.1 3APL

3APL (An Abstract Agent Programming Language) (Hindriks 1999, Dastani 2004) is a programming language for implementing cognitive agents. Agents written in 3APL language consist of goals, belief, practical reasoning rules and capabilities. A goal is a state of the system that the agent wants to achieve. A Belief is used to represent the current mental state of the agent. Practical reasoning rules are the means for the agent to manipulate the goals. Capabilities are the actions that can be performed by the agent. An action can only be performed if certain beliefs hold, this is called precondition of an action.

In this paper, we adopt 3APL platform (Dastani 2004) to support our work. Our work is mainly based on 3APL definitions from (Hindriks 1999, Dastani 2004).

Definition 1 A 3APL agent is defined as a tuple 〈n, B, G, P, A〉, where n is the name of the agent, B is a set of beliefs (Beliefbase), G is a set of goals (Goalbase), P is a set of practical reasoning rules (Rulebase) and A is a set of basic actions (Capabilities).

In (Hindriks 1999), a set of programming constructs for goals are defined, namely *BactionGoal*, *PreGoal*, *TestGoal*, *SkipGoal*, *SequenceGoal*, *IfGoal*, *WhileGoal* and *JavaGoal*, which can be used in the body part of a practical reasoning rule and make 3APL more flexible.

In a 3APL agent, P is a set of rules in the form:

π_h <-φ | π_b,

In this formula, π_h and π_b belong to a goal variable set, and φ is a belief. When the agent has goal π_h and believes φ then π_h is replaced by π_b.

For a 3APL agent, Beliefbase is dynamic. It is updated with executing basic actions from the set of capabilities. The structure of a basic action is shown below:

$\{\varphi_1\}$ Action(X) $\{\varphi_2\}$

φ_1 is the pre-condition and φ_2 is the post-condition. Precondition and post-condition are belief formulas. It is possible to have an action that does not have any pre-condition or post condition. The execution of an action will result in the update of beliefbase through replacing preconditions by postconditions. The beliefbase can also be extended with a Prolog program (facts and rules) using the LOAD option (Dastani 2004). In addition, beliefs can be generated from the communications between two agents (sent and received). 3APL has a mechanism to support the communications between agents. A message mechanism is defined in (Dastani 2004) to fulfill the communication between agents. The messages themselves have a specific structure, *Receiver/ Sender*, *Performative* are three compulsory elements in a message. Usually, there are three type of message: *send*(*Receiver*, *Performative*, *Content*)*, sent*(*Receiver*, *Performative*, *Content*), *and received*(*Sender*, *Performative*, *Content*). This agent communication mechanism is described in details in (Dastani 2004).

In this paper we shall not elaborate more on the syntax of 3APL, readers who may want more details are directed to (Hindriks 1999, Dastani 2004).

3.2 Mapping i* Model to 3APL Agents

We view an *i** model as a pair 〈SD, SR〉 where SD is a graph denoted by 〈*Actors*, *Dependencies*〉 where *Actors* is a set of nodes (one for each actor) and *Dependencies* is a set of labeled edges. These edges can be of 4 kinds: *goal dependencies*(denoted by D_G(SD))*, task dependencies*(denoted by D_T(SD))*, resource dependencies*(denoted by D_R(SD)) and *softgoal dependencies*(denoted by D_S(SD)). Each edge is defined as a triple 〈T_o, T_d, ID〉, where T_o denotes the *depender*, T_d denotes the *dependum* and ID is the label on the edge that serves as a unique name and includes information to indicate which of the four kinds of dependencies that edge represents. SR is a set of graphs, each of which describes an actor.

We adopt the concept of an environment simulator agent (ESA) defined in (Salim 2005).

We define MAS is a pair ⟨*Agents*, *ESA*⟩ where *Agents* = {a_1, ..., a_n}, each a_i is a 3APL agent and *ESA* is a specially designated Environment Simulator Agent implemented in 3APL which holds the knowledge about the actions that might be performed by actors in SD model and the possible environment transformation after the executions of those actions. The environment agent can verify fulfilment properties (clearly defined in Formal Tropos (Fuxman 2004)), which include conditions such as *creation conditions, invariant conditions, and fulfillment conditions* of those actions associated with each agent. Every action of each agent has those fulfilment properties. *ESA* is used to check whether those actions of all agents in this system satisfy corresponding conditions.

Each graph in an SR model is a triple⟨*SR-nodes*, *SR-edges*, *ActorID*⟩. The *SR-nodes* consist of a set of goal nodes (denoted by N_G), a set of task nodes (denoted by N_T), a set of resource nodes (denoted by N_R) and a set of softgoal nodes (denoted by N_S). *SR-edges* can be of 3 kinds: means-ends links (denoted by the set *MELinks*), task-decomposition link (denoted by the set *TDLinks*) and softgoal contribution link (denoted by the set *SCLinks*). Each *MELink* and *TDLink* is represented as a pair, where the first element is the parent node and the second element is the child node. A *SCLink* is represented as a triple⟨*s*, *m*, *c*⟩, where the first element is the parent node, the second element is the child node and the third element is the *softgoal contribution* which can be *positive* or *negative*.

Any MAS ⟨*Agents*, *ESA*⟩ obtained from an *i** model *m*=⟨*SD*, *SR*⟩, where *SD*=⟨*Actors*, *Dependencies*⟩ and *SR* is a set of triples of the form⟨*SR-nodes*, *SR-edges*, *ActorID*⟩ (we assume that a such a triple exists for each actor in *Actors)* with *SR-nodes*= $N_G \cup N_T \cup N_R \cup N_s$ and *SR-edges*=*MELinks* ∪ *TDLinks* ∪ *SCLinks* must satisfy the following conditions (Guan 2005):

1. For all *a* in *Actors*, there exists an agent in *Agents* with the same name.

For example, in the Online Shopping System, *Retail system* is an actor in SR Model, therefore, there is an agent named "Retail System" in this 3APL agents system.

2. For every goal node or task node *n* in the SR diagram for that actor, the corresponding agent ⟨*a*, *B*, *G*, *P*, *A*⟩ in *Agents* must satisfy the property that *goal*(*n*) or *task*(*n*) in the *G*.

For example, goal *Sell product* and task *Handle Online Order* are in the boundary of actor *Retail System*, according to step 2, *SellProduct*() and *HandelOnlinOrder*() are in the goalbase of agent *RetailSystem*.

3. For all *a* in *Actors* and for each *p* in N_G (parent node) for which a link ⟨*p*, *c*⟩ in *MELink* exists in the SR model for that actor, with *c* in N_T (children node), the corresponding agent ⟨*a*, *B*, *G*, *P*, *A*⟩ in *Agents* must satisfy the property that *goal*(*p*)<- φ | *SeqComp*(*T*) is an element of *P*. Here $T=\{c_1,\ldots,c_n\}$, given that $<p,c_1>,\ldots,<p, c_n>$ are all the task decomposition links that share the same parent *p*. *SeqComp*(*T*) is an operation that generates the body of the practical reasoning rule referred to above by sequentially composing the goal or task children identified in each of the means-end links with the same parent *p*. The *i** model in itself does not provide any information on what this sequence should be. This needs to be provided by the analyst or, by default, obtained from a left-to-right reading of the means-ends-links for the same parent in an SR diagram.

For example, in the SR diagram of actor *Retail System* of figure 1, task *Handle Online Order* and goal *Sell Product* are connected by a means-end link, therefore, rule *SellProduct*()<-φ | *HandelOnlineOrder*() can be added into the Rulebase of agent *RetailSystem.* Belief fomula φ and parameters of goal and task can be specified according to the real case.

4. For all *a* in *Actors* and for each *p* in N_T for which a link ⟨*p*, *c*⟩ in *TDLink* exists in the SR model for that actor (where *c* in ($N_T \cup N_G$)), the corresponding agent ⟨*a*, *B*, *G*, *P*, *A*⟩ in *Agents* must satisfy the property that *goal*(*p*)<- φ| *SeqComp*(*T*) is an element of *P*. Here $T=\{c_1,\ldots,c_n\}$, given that $<p,c_1>,\ldots,<p, c_n>$ are all the task decomposition links that share the same parent *p*. *SeqComp*(*T*) is as defined in rule 3.

Note that, in the rules defined above, the execution orders of sub-tasks within the *Task-decomposition links* are from left to right as default. Belief formulas of each practical reasoning rule cannot be generated completely automatically, instead those beliefs are specified by the designers.

Take task *Handel Online Order* as the parent task node for example, this task is decomposed into three sub-tasks: *Confirm Customer*, *Let Payment System Handel Payment* and *Let Product Management System Send Product*. Using the above rule, will lead to:

```
    HandleOnlineOrder() <- φ |
      BEGIN

letpaymentsystemhandelpayment();
        confirmcustomer();

letproductmanagementsystemsendproduct()
      END.
```

Note that, in the rules defined above, the execution orders of sub-tasks within the *Task-*

decomposition links are from left to right as default. Belief formulas of each practical reasoning rule cannot be generated completely automatically; instead, those beliefs are specified by designers.

5. For all *a* in *Actors* and for each triple⟨*s*, *m*, *c*⟩in *SCLinks* in the SR model for that actor, the corresponding agent ⟨*a*, *B*, *G*, *P*, *A*⟩in *Agents* must satisfy the property that *belief*(*m*, *s*, *c*) is an element of *B*. We do not describe how beliefs about softgoal contributions are used in agent programs for brevity – we will flag however that they can plan a critical role in selecting amongst practical reasoning rules.

For example, there are two ways to achieve goal *Own Product* for an actor, one is *Go Shopping*, the other is *Shopping Online*. On the assumption that task *GoShopping has positive contribution to softgoals low effort*, *convenient* and *time saving* while task *ShoppingOnline has positive effects on those three softgoals.*

The following beliefs are in the beliefbase of agent *Customer*.

```
Belief(OwnProduct, GoShopping,
timesavingnegative).
Belief(OwnProduct, GoShopping,
loweffortnegative).
Belief(OwnProduct, GoShopping,
convenientnegative).
Belief(OwnProduct, ShoppingOnline,
timesavingpositive).
Belief(OwnProduct, ShoppingOnline,
loweffortpositive).
Belief(OwnProduct,ShoppingOnline,
convenientpositive).
```

6. For all dependencies⟨T_o, T_d, ID⟩ in SD, there exist agents ⟨T_o, B_o, G_o, P_o, A_o⟩,⟨T_d, B_d, G_d, P_d, A_d⟩in Agents, such that if ⟨T_o, T_d, ID⟩ is in D_G(SD), then goal(ID) is an element of G_o,

```
goal(ID) <-φ|
  BEGIN
      send(Td, request,
requestAchieve(ID));
      send(ESA, inform,
believe(φ))
  END  is an element of  Po,
  received(To, request,
requestAcheive(ID)) |
  BEGIN
     Achieve(ID);
    send(ESA, inform,
believe(Achieved(ID))
  END is an element of  Pd.
```

Here φ denotes the *creation condition* of the dependency *ID*.

Similarly, if ⟨T_o, T_d, *ID*⟩ is in D_T*(SD)* , *task*(*ID*) is in G_o,

```
Task(ID) <-φ |
  BEGIN
    send(Td, request,
requestPerform(ID));
    send(ESA, inform
,believe(φ))
  END ∈ Po
  received(To, request,
requestPerform (ID)) |
  BEGIN
    Perform(ID);
   send(ESA, inform,
believe(Performed(ID))
  END  is an element of  Pd.
```

Similarly, if ⟨T_o, T_d, *ID*⟩ is in D_R(*SD*) then

```
Request(ID) <-φ |
  BEGIN
    send(Td, request,
requestProvide(ID));
   send(ESA, inform ,believe(φ))
  END ∈ Po,
  received(To, request,
requestProvide(ID)) |
  BEGIN
   send(To,request,offer(ID));
  send(ESA, inform,
believe(Offered(ID))
  END is an element of  Pd.
```

Notice that these rules require that the creation conditions be communicated by the depender agent to the ESA agent. The ESA monitors all of the actions/tasks performed by each agent, all of the messages exchanged and all of the beliefs (usually creation conditions for dependencies) communicated by individual agents for consistency and for constraint violations (e.g. the FormalTROPOS-style conditions associated with dependencies). When any of these is detected, the ESA generates a user *alert.*

We shall select one task-dependency and one resource-dependency related to agent *Retail* System in order to illustrate rule 6. Actor *Customer* depends on actor *Retail System* to perform task *Buy Product Online* and to provide *Confirmation* of buying. According to rule 6, for agent *Customer*, *BuyProductOnline*() is in the Goalbase. Rules shown below are in its Rulebase:

```
Request(confirmation) <-
   product(P) AND
needconfirmation(P) |
  BEGIN
    send(retailsystem, request,
requestProvide(confirmation));
    send(ESA, inform
,believe(needconfirmation))
  END
Task(BuyProductOnline) <-
  needtobuyproductonline |
```

```
      BEGIN
        send(retailsystem, request,
requestPerform(BuyProductOnline));
        send(ESA, inform ,believe(φ))
END are in Rulebase.
   For agent RetailSytem, two rules are
generated for these two dependencies
relationships.
   received(customer, request,
requestProvide(confirmation)) |
      BEGIN
        send(customer, request,
offer(confirmation));
        send(ESA, inform,
believe(Offered(confirmation))
      END
   received(customer, request,
requestPerform (BuyProductOnline)) |
      BEGIN
        Perform(BuyProductOnline);
        send(ESA, inform,
believe(Performed(BuyProductOnline))
      END
```

Figure 2 (provided below) is a snapshot for the Online Shopping 3APL agent System. It provides insight into the communication messages.

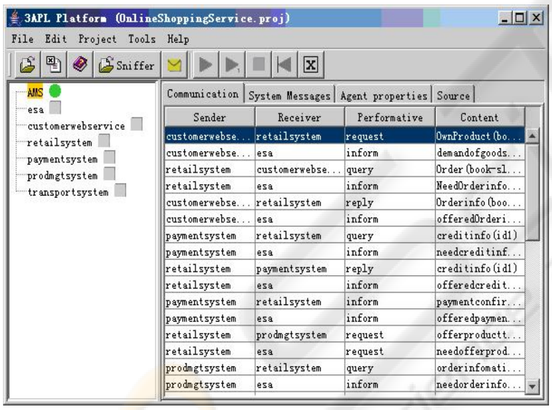

Figure 2: Communication messages.

4 CONCLUSIONS

This paper has shown how agent-oriented conceptual modelling techniques such as i* framework can be employed to model service-oriented systems. Along with this we have suggested an approach to executing i* models by translating these into a set of interacting agents (services) implemented in the 3APL language. This approach makes use of the advantages of *i** for the early-phase of requirement engineering and validates the model by mapping it into an executable specification to see the design result in an emulation program. We are working towards automating the approach proposed in this paper.

REFERENCES

Castro, J., Kolp, M., Mylopoulos, J., (2002) Towards requirements-driven information systems engineering: the Tropos project, *Information Systems Journal, 2002.*

Chen, F., Yang, H., Guo, H., Xu, B. (2004) Agentification for web services, *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 2004 pp.514 – 519

Chinnici, R. (2002) Web Service Description Language (WSDL) Version 1.2, World Wide Web Consortium, 2002, www.w3.org/TR/wsdl12/

Chung, L., Nixon, B., Yu, E., Mylopoulos, J. (2000) *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishing, 2000

Dastani, M. (2004) 3APL Platform User Guide November 16, *Utrecht University*, 2004

Diana, L., Mylopoulos, M. (2004) Designing Web Services with Tropos, *Proceedings of ICWS'04*, San Diego, USA, 2004. IEEE Computer Society Press

Fuxman, A., Liu, L., Pistore, M., Roveri, M., Mylopoulos, J. (2004) Specifying and Analyzing Early Requirements in Tropos, Requirements Engineering Journal, 9(2): 132-150, 2004

Guan, Y., Ghose, A. K. (2005) Executable specifications for agent-oriented conceptual modelling. *Proceedings of Proceedings of the IEEE/WIC 2005 International Conference on Intelligent Agent Technology*, France, 2005

Hindriks, K. V., De Boer, F. S., Van der, H. W., Meyer, J. (1999) Agent programming in 3APL. *Autonomous Agents & Multi-Agent Systems*, 2(4):357- 401, 1999

Jørstad, I., Dustdar, S., van Do, T. (2005). A Service Oriented Architecture Framework for collaborative services. *3rd International Workshop on Distributed and Mobile collaboration (DMC), IEEE WETICE*, Linköping, Sweden, 13-15 June 2005

Lohse, G., Spiller, P. (1998) "Electronic shopping", *Communications of the ACM*, 41(7): 81–87, July 1998

Salim, F., Chang, C., Krishna, A., Ghose, A. K. (2005) Towards executable specifications: Combining i* and AgentSpeak (L). *Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering (SEKE-2005)*, Taipei, Taiwan, July, 2005

Universal Description, Discovery and Integration, Organization for Advancement of Structured Information System, (2002), www.uddi.org/specification.html

Yu, E. (1995) Modelling Strategic Relationships for Process Reengineering. *PhD Thesis, Graduate Department of Computer Science, University of Toronto*, Toronto, Canada, 1995, pp. 124