Agent-Based Prototyping of Web-Based Systems

Aneesh Krishna, Ying Guan, Chattrakul Sombattheera, and Aditya K. Ghose

Decision Systems Laboratory, School of IT and Computer Science
University of Wollongong, NSW 2522, Australia
{aneesh, yg32, cs50, aditya}@uow.edu.au

Abstract. Agent-oriented conceptual modelling in notations such as the *i** framework [13] have gained considerable currency in the recent past. Such notations model organizational context and offer high-level social/ anthropomorphic abstractions (such as goals, tasks, softgoals and dependencies) as modelling constructs. It has been argued that such notations help answer questions such as what goals exist, how key actors depend on each other and what alternatives must be considered. Our contribution in this paper is to show an approach to executing high-level requirements models represented in the *i** agent-oriented conceptual modelling language. We achieve this by translating these models into sets of interacting agents implemented in the 3APL language. This approach enables us to analyze early phase system models by performing rule-/consistency-checking at higher-levels of abstraction. We show how this approach finds special application in the analysis of high-level models of a web-based system.

1 Introduction

Web-based systems are playing a more and more important role in the modern society nowadays. Based on the persistent nature and the colossal user population, a variety of existing software engineering approaches need to be adapted for web engineering. The design and development of a web-based system may include many business, technology and user challenges. Understanding and satisfying the critical requirements of these challenges will decide the success of a web-based systems commercial application. Web-based systems are more scalable and available to remote users, enabling utilities to better support decentralized operations during high-activity periods. A web-based system is a form of distributed systems architecture, essentially a collection of services. Web based system provides consistent interoperability and reuses existing services where possible. Implementing a web-based system can involve developing applications that use services and making applications available as services. 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 [3] and UDDI [12]. Web services are built on the distributed environment of the Internet.

Agents are components that aim to communicate models motivated from real life [2]. The development of agent-based systems offers a new and existing paradigm for the creation of sophisticated programs in a dynamic and open environment, particularly in distributed domains such as a web-based systems and electronic commerce [10].

© Springer-Verlag Berlin Heidelberg 2006

Current works on web services are closely entwined with work on agent-based systems. Agent-oriented techniques show a potential for web services, where agents are needed both to offer services and to make best use of the resources available [2]. 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 [13] 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 create 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 [4] 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 (exemplified by the *i** framework) can be used to model web-based systems, and how these models can be executed by mapping *i** models into 3APL [9] 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 web-based systems are given. We shall provide the executable specifications for the *i** framework in section 3. Section 3 also provides an example to illustrate the approach and section 4 presents some concluding remarks.

2 Modelling Web-Based Systems

Most of the available modelling techniques tend to address “late-phase” requirements while the vast majority of critical modelling decisions are arguably taken in early-phase requirements engineering [13]. 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 [13] for details on *i** framework.

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 [6], the authors have proposed an approach based on the Tropos methodology [1], for designing web services. Our proposal is different from theirs in the sense that, we focus on to modeling web-based systems in the early requirement phase, and validate these models by executable specifications, while in [6], 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. The modelling process includes following steps.

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: After identifying the actors of a web service, their goals are defined simultaneously. In this case, the actor Customer/ Web server has the goal own product online and the 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 (optimization objectives or preferences) to be satisfied [6]. 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 fulfill 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.

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. For example, the Customer/Web Server actor has an internal task to perform ShoppingOnline. This task can be performed by subtasks SelectProduct and PlaceOrder (these are related to the parent task via task decomposition links). The SelectProduct task is further decomposed into subtasks BrowseCatalog and SearchProduct.

Following the four steps that were mentioned, 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 Mapping *i** Model to 3APL Agents

3APL (An Abstract Agent Programming Language) [9] [5] 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 [5] to support our work. Our work is mainly based on 3APL definitions from [9] [5].

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 [5], 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.

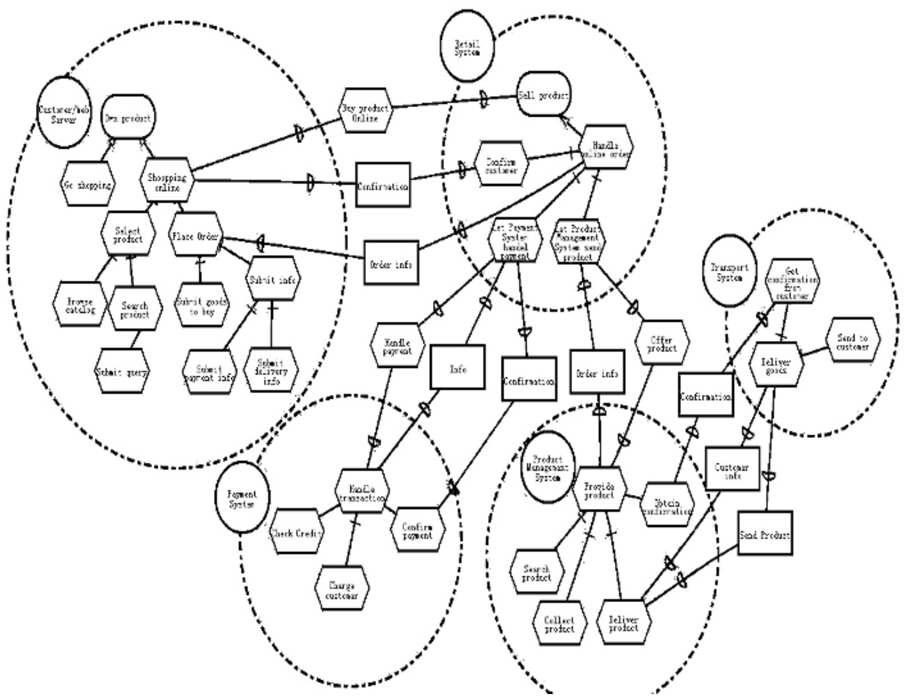


Fig. 1. Strategic Rationale Model of Online shopping service

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 [5]. 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 [5] 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 [5]. In this paper we will not elaborate more on the syntax of 3APL, readers who may want more details are directed to [9] [5].

We view an *i** model as a pair $\langle$ SD, SR $\rangle$ where SD is a graph denoted by $\langle$ *Actors*, *Dependencies* $\rangle$ 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 $\langle T_o, T_d, \mathrm{ID} \rangle$, 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 [11].

We define MAS is a pair $\langle$ *Agents*, *ESA* $\rangle$ 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 fulfillment properties (clearly defined in Formal Tropos [7]), 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 fulfillment 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 $\langle$ *SR-nodes*, *SR-edges*, *ActorID* $\rangle$. 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 $\langle s, m, c \rangle$, 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 $\langle$ *Agents*, *ESA* $\rangle$ obtained from an *i** model $m = \langle SD, SR \rangle$, where $SD = \langle$ *Actors*, *Dependencies* $\rangle$ and *SR* is a set of triples of the form $\langle$ *SR-nodes*, *SR-edges*, *ActorID* $\rangle$ (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* $\cup$ *TDLinks* $\cup$ *SCLinks* must satisfy the following conditions [8]:

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 $\langle a, B, G, P, A \rangle$ 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 $\langle p, c \rangle$ in *MELink* exists in the SR model for that actor, with *c* in N_T (children node), the corresponding agent $\langle a, B, G, P, A \rangle$ 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 procedural reasoning rule referred to above by sequentially composing the goal or task children identified in each of the means-ends 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 formula 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 $\langle p, c \rangle$ in *TDLink* exists in the SR model for that actor (where *c* in ($N_T \cup N_G$)), the corresponding agent $\langle a, B, G, P, A \rangle$ 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 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 Handle 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 $\langle s, m, c \rangle$ in *SCLinks* in the SR model for that actor, the corresponding agent $\langle a, B, G, P, A \rangle$ 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 procedural 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.*
6. For all dependencies $\langle T_o, T_d, ID \rangle$ in SD, there exist agents $\langle T_o, B_o, G_o, P_o, A_o \rangle$, $\langle T_d, B_d, G_d, P_d, A_d \rangle$ in Agents, such that if $\langle T_o, T_d, ID \rangle$ is in D_G(SD), then goal(ID) is an element of G_o,

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
```

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*

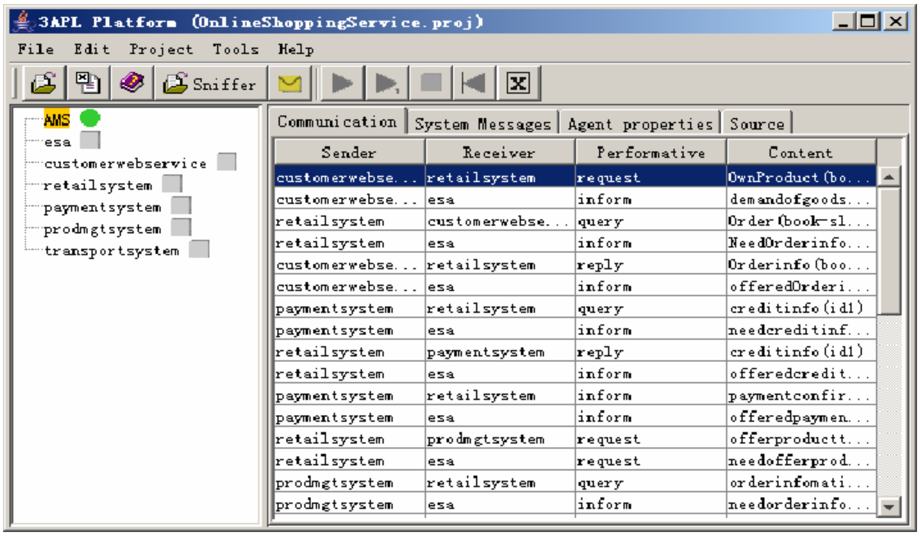
System to perform task *Buy Product Online* and to provide *Confirmation* of buying. According to rule 6, for agent *Customer*, *BuyProductOnline*() is in the Goalbase. Rules shown below are in its Rulebase:

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

Fig. 2 (provided below) is a snapshot for the Online Shopping 3APL agent System. It provides insight into the communication messages and flow of the desired system.



Sender	Receiver	Performative	Content
customerwebse...	retailsystem	request	OwnProduct(bo...
customerwebse...	esa	inform	demandofgoods...
retailsystem	customerwebse...	query	Order(book-sl...
retailsystem	esa	inform	NeedOrderinfo...
customerwebse...	retailsystem	reply	Orderinfo(boo...
customerwebse...	esa	inform	offeredOrderi...
paymentsystem	retailsystem	query	creditinfo(id1)
paymentsystem	esa	inform	needcreditinf...
retailsystem	paymentsystem	reply	creditinfo(id1)
retailsystem	esa	inform	offeredcredit...
paymentsystem	retailsystem	inform	paymentconfir...
paymentsystem	esa	inform	offeredpaymen...
retailsystem	prodmgtsystem	request	offerproductt...
retailsystem	esa	request	needofferprod...
prodmgtsystem	retailsystem	query	orderinfomati...
prodmgtsystem	esa	inform	needorderinfo...

Fig. 2. Communication messages

4 Conclusions

This paper has shown how agent-oriented conceptual modelling techniques such as *i** framework can be employed to model web-based 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. Furthermore, we also believe in the co-evolution approach in which models are composed of *i** model and 3APL agents. How to co-evolve *i** model and 3APL agents remains for future works.

References

1. Castro, J., Kolp, M., Mylopoulos, J., Towards requirements-driven information systems engineering: the Tropos project, *Information Systems Journal*, 2002, 27(6), pp. 365-389
2. Chen, F., Yang, H., Guo, H., Xu, B. Agentification for web services, *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 2004, pp. 514 – 519
3. Chinnici, R. Web Service Description Language (WSDL) Version 1.2, World Wide Web Consortium, 2002, www.w3.org/TR/wsdl12/
4. Chung, L., Nixon, B., Yu, E., Mylopoulos, J. *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishing, 2000
5. Dastani, M. 3APL Platform User Guide November 16, *Utrecht University*, 2004
6. Diana, L., Mylopoulos, M. Designing Web Services with Tropos, *Proceedings of ICWS'04*, San Diego, USA, 2004. IEEE Computer Society Press
7. Fuxman, A., Liu, L., Pistore, M., Roveri, M., Mylopoulos, J. Specifying and Analyzing Early Requirements in Tropos, *Requirements Engineering Journal*, 2004, 9(2), pp. 132-150
8. Guan, Y., Ghose, A. K. Executable specifications for agent-oriented conceptual modeling. *Proceedings of the IEEE/WIC 2005 International Conference on Intelligent Agent Technology*, France, 2005
9. Hindriks, K. V., De Boer, F. S., Van der, H. W., Meyer, J. Agent programming in 3APL. *Autonomous Agents & Multi-Agent Systems*, 1999, 2(4), pp. 357- 401
10. Lohse, G., Spiller, P. (1998) "Electronic shopping", *Communications of the ACM*, July 1998, 41(7), pp. 81–87
11. Salim, F., Chang, C., Krishna, A., Ghose, A. K. 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
12. *Universal Description, Discovery and Integration*, Organization for Advancement of Structured Information System, (2002), www.uddi.org/specification.html
13. Yu, E. Modelling Strategic Relationships for Process Reengineering. *PhD Thesis, Graduate Department of Computer Science, University of Toronto, Toronto*, Canada, 1995, pp. 124