An Open Electronic Marketplace through Agent-based Workflows: MOPPET[1]

SENA ARPINAR, ASUMAN DOGAC AND NESIME TATBUL

Software Research and Development Center

Dept. of Computer Engineering, Middle East Technical University (METU)

06531 Ankara, Turkey

{nural,asuman,tatbul}@srdc.metu.edu.tr

Abstract

We propose an electronic marketplace architecture, called MOPPET, where the commerce processes in the marketplace are modeled as adaptable agent-based workflows. The higher level of abstraction provided by the workflow technology makes the customization of electronic commerce processes for different users possible. Agent-based implementation, on the other hand, provides for a highly reusable component-based workflow architecture as well as negotiation ability and the capability to adapt to dynamic changes in the environment. Agent communication is handled through Knowledge Query and Manipulation Language (KQML). A workflow-based architecture also makes it possible for complete modeling of electronic commerce processes by allowing involved parties to be able to invoke already existing applications or to define new tasks and to re-structure the control and data flow among the tasks to create custom built process definitions.

In the proposed architecture all data exchanges are realized through Extensible Markup Language (XML) providing uniformity, simplicity and a highly open and interoperable architecture. Metadata of activities are expressed through Resource Description Framework (RDF). Common Business Library (CBL) is used for achieving interoperability across business domains and domain specific Document Type Definitions (DTDs) are used for vertical industries. We provide our own specifications for missing DTDs to be replaced by the original specifications when they become available.

Keywords: *electronic marketplace, workflow, agent, Extensible Markup Language (XML), Common Business Library (CBL).*

---

[1] This work is being partially supported by Middle East Technical University, Project Number: AFP-97-07-02-08 and by the Scientific and Technical Research Council of Turkey, Project Number: 197E038.

Introduction

Markets play a central role in the economy, facilitating the exchange of information, goods, services, and payments. They have three main functions: matching buyers and sellers; facilitating the exchange of information, goods, services and payment associated with market transactions; and providing an institutional infrastructure, such as legal and regulatory framework, that enables the efficient functioning of the market [Y. Bakos, 1998].

Recent years have seen a dramatic increase in the role of information technology in markets, both in traditional markets, and in the emergence of electronic marketplaces, such as the maltitute of Internet-based online auctions. "eBay" (http://www.ebay.com) is an example of a very successful marketplace where over 2 million items are being auctioned in more than 1500 categories. Other examples include “Bargain Finder” (http://bf.cstar.ac.com), developed by Anderson Consulting as part of their Smart Store Virtual iniative, and "FireFly" (http://www.agents-inc.com), from Agents Inc.

In spite of a quite a number of successful examples, Internet-based electronic marketplaces are still at a formative stage. An open interoprable platform exploiting the emerging standards and technologies is not there yet; the agent technology has not been fully exploited and developed to cope with tremendous amount of information available; the processes involved in electronic commerce have not yet been automated to a desirable extent; and the services a marketplace offers to its customers need to be improved.

In this paper, we propose an electronic marketplace architecture, called MOPPET (METU OPen Electronic MarkeTplace) to address these issues. The features offered by MOPPET are as follows:

1. Electronic commerce processes in the marketplace are modeled as workflow processes, which are realized through agent-based components.

   Many researchers as well as commercial companies have created agent-based systems that support various aspects of electronic commerce such as online

shopping, virtual catalogs or electronic marketplaces. While these systems provide interesting shopping experiences, they fall short in fully exploiting the capabilities offered by the electronic medium.

These systems can not handle diversity of customer needs. As an example, a customer may want to buy more than one related item and there can be dependencies among the items and also compatibility requirements that stem from the nature of these items. For example, s/he may want to buy a printer together with a personal computer. This creates a dependency between the computer and the printer. Also, for the specific software that s/he considers there could be a certain amount of memory requirement. This illustrates a compatibility requirement. In contrast, current systems are mostly designed to handle one request at a time. For instance, a customer may buy an item by searching several shops/stores but can not make several inquiries in one step to buy several compatible and related items and/or services. Shopping carts support several inquiries of a buyer however the buyer cannot give dependencies among the items or the specific order of the related purchases. Also, no support is provided for compatibility requirements. In our approach, the dependencies expressed by the user are represented through control flow dependencies. For expressing the compatibility requirements among items, there is a need for a knowledge base to store the rules.

More importantly, most of the systems developed do not have enough facilities to automate the business processes conducted by the user/customer. In this respect, we propose to organize the electronic commerce processes into workflow templates adaptable to user needs. Workflow based approach allows involved parties to define their own tasks and to invoke already existing applications within the workflow and to re-structure the control and data-flow among the tasks, in other words, to automatically create a custom built workflow from the workflow template. The higher level of abstraction provided by the workflow technology makes this customization of processes for different users possible. In addition, the workflow definition makes it possible to invoke any number of activities in parallel to provide efficiency and to dynamically reengineer the commerce processes not only to the user

needs but also to balance the system workload. For example, the search and purchase of related items from different stores can be activated in parallel. Furthermore the recovery functionality of a workflow system allows to automatically rollback the necessary activities if a dependent activity fails, e.g., a desk purchase request of a user executing in parallel with his computer purchase request can be automatically rolled back if the computer purchase request fails.

The workflow system architecture is designed to consist of functionality based reusable components each of which is realized through different types of agents. Use of agents provides greater flexibility, agility and adaptibility especially due to their properties of being proactive and responsive. They are proactive in the sense that they can take the initiative when an unaniticipated condition occurs and are responsive so that they can sense and respond the changes in the environment. Negotiation ability of agents provides for another aspect of flexibility in the execution of workflow processes.

At the lowest level, there is a need to invoke different types of tasks. To achieve this functionality, specific *task agents* are designed which can be reused whenever the need arises. There are task agents for querying the XML documents, for negotiation and for handling activities requiring user attention. The scheduler of a workflow determines the possible control and data-flow among task agents. There could be modifications on the control-flow depending on the negotiation among the agents at run-time. This scheduling functionality is also designed as an agent since the scheduler needs to adapt to dynamic changes in the environment and also may need to negotiate with other scheduling agents for delegating parts of a workflow process. Such a *scheduling agent* implemented according to a workflow DTD is a highly reusable component, since it can enact any workflow definition written in XML conforming to this DTD. It also gives the user the flexibility to include any process definition conforming to workflow DTD in the workflow template. *Recovery* component takes the initiative when a failure or an unanticipated change in the specification occurs and is realized as an agent. Another type of agent is *facilitator agent* whose responsibility is to allow

agents to find each other (through advertisements) and to provide services such as a naming service. Interactions with the user are handled by an agent called *interface agent* so that the underlying complexity of the system is hidden from the user. Interface agents provide graphical user interfaces to their users and get the requirements of the users to compose workflow process definitions by adapting the existing templates.

2. The interoperability infrastructure is based on XML.
   For electronic commerce to become really ubiquitous electronic commerce architectures should be open, that is, they should be based on infrastructures providing for semantic interoperability. The most promising proposal in providing an open and interoperable electronic commerce architecture seems to be the efforts of World Wide Web Consortium (W3C) in providing data exchange and data semantic standards like XML, RDF and CommerceNet's efforts on developing an open electronic commerce framework based on Common Business Library (CBL).

   XML has gained a great momentum and is emerging as the standard for self-describing data exchange on the Internet. Its power lies in its extensibility and ubiquity. Anyone can invent new tags for particular subject areas and define what they mean in document type definitions. Content oriented tagging enables a computer to understand the meaning of data. But if every business uses its own XML definition for describing its data, it is not possible to achieve interoperability. The tags need to be semantically consistent across merchant boundaries. For this reason, CBL which consists of product taxonomies and message formats as XML DTDs, has been developed [B. Meltzer and R. Glushko, 1998] and the baseline version (1.1) is available from [VEO Systems Inc., 1998]. CBL contains a set of building blocks common to many business domains such as address (location.dtd), price (value.dtd), purchase order (order.dtd) and standard measurements (measures.dtd), and thus provides the much-needed basis to ensure interoperability among XML applications. When this is complemented by a set of DTDs common for specific industries, that is for vertical domains, the open electronic commerce infrastructure will be achieved. In fact, some of the specifications for vertical

domains are already available like HL7 for exchanging healthcare records, OBI (Open Buying on the Internet), OTP (Open Trading Protocol), and work is going on for some other domains like for producing the common terminology and structure of documents for personal computers [RosettaNet, 1998]. The interoperability infrastructure of MOPPET makes use of CBL and DTDs for vertical domains. We provide our own specifications for missing DTDs to be replaced by the originals when they become available. In this respect, we provide DTDs such as a DTD for workflow definition (*workflow.dtd*). The advantage of defining a workflow as a DTD is that as long as there is a workflow engine on the Internet that can interpret this *workflow.dtd*, any workflow process defined in XML conforming to this DTD can be executed. This gives an enormous flexibility in terms of interoperability. A further level of interoperability is necessary for systems involving agents. For this purpose, we use Knowledge Query and Manipulation Language (KQML) which is a language and a protocol for exchanging information and knowledge among agents. Since KQML is designed for knowledge sharing, we extend it for agent negotiation.

The organization of the paper is as follows: Following section presents our agent-based workflow management system architecture. In the next section, we provide an electronic marketplace environment, called MOPPET. Next, we explain the proposed architecture providing a detailed scenario. We, then, describe related work for developing marketplaces and also some previous work on building agent-based workflow management systems. Final section includes conclusions.

Agent-based WfMS Architecture

In this section, a workflow management system (WfMS) architecture that is used as a building block in the MOPPET marketplace is described.

Workflow management deals with the specification and execution of business processes. Workflow management systems allow one to define, execute, manage, and modify business processes. Business processes, especially for electronic commerce are highly dynamic and unpredictable - it is difficult to give a complete

priori specification of all activities that need to be performed and how they should be ordered. In addition, a workflow system must include a flexible enactment system that is capable of supporting scalability, where new resources can be incorporated easily within the workflow system; and adaptive workflows, where the workflow specification can be changed or extended. With this in mind, electronic commerce processes in the marketplace are modeled through agent-based components. In this way, it is possible to partition a potentially large load among participating components, ie., agents and to guarantee minimal communication between these components which are essential to achieve scalability where a workflow system may need to support tens of thousands of active order processing workflow instances such as in an electronic marketplace.

There are various agent-based workflow systems developed as explained in related work section. However, they lack the means for interoperating with other workflow systems and custom-built services. In order to achieve this level of interoperability, we make use of standardization efforts like KQML [Y. Labrou and T. Finin, 1997] and XML [XML, 1998].

The Extensible Markup Language (XML) is a data format for structured document interchange on the Web. It provides a framework for tagging structured data by allowing developers to define an unlimited set of tags bringing great flexibility. XML resembles and complements HTML. XML describes data such as city name, temperature, and HTML defines tags that describe how the data should be displayed such as with a bulleted list or a table. Document Type Definitions (DTDs) may accompany an XML document, essentially defining the rules of document, such as which elements are present and the structural relationships between the elements. DTDs help to validate the data. XML brings so much power and flexibility to Web-based applications that it provides a number of benefits to developers such as being able to do more meaningful searches.

Agents for Workflow Process Enactment

In our architecture, workflow process enactment is performed by means of agents that exchange XML through KQML messages. In other words, all data and definitions are written in XML such as workflow definition and all agents in the system uses KQML for communicating with other agents.

It should be noted that, KIF (Knowledge Interchange format) might also be used in KQML communications among agents but we prefer using RDF and XML due to their power to lead more open and easy to understand architecture. XML is more restricted than RDF since it only uses hierarchical representation of data but in most circumstances in MOPPET, it is found to be sufficient to use XML and in cases it is not adequate such as describing roles, product taxonomies or related product information, agents exchange documents in RDF.

There are five types of agents in the system: interface agents, scheduling agents, task agents, recovery agents and facilitator agents. In the following each agent's functionality is explained.

Interface Agent. Interface agents are responsible for collecting and collating relevant information from the user to initiate a workflow process, presenting the returned results and explanations to the user, requesting the user for additional information during recovery and asking for user confirmation when necessary. By means of interface agents, the complexity and underlying distribution are hidden from the user. After getting initial specification from the user, interface agent constructs the initial schedule of tasks needed to satisfy user specification. Then, it finds and contacts to a scheduling agent to execute the workflow through negotiation. The negotiation strategy employed by agents is explained at the end of this section.

Scheduling Agent. The scheduling agent is the agent doing the real workflow enactment. It gets the workflow definition in XML from the interface agent of the user and tries to execute all the tasks and subprocesses according to the control and data-flow given in the definition. It contacts task agents to schedule the individual tasks or may contact other scheduling agents to delegate some part of

the workflow. To find the relevant task/scheduling agent, it negotiates with possible set of agents and it schedules the job to one of them. One of the other responsibilities of the scheduling agent is to ensure the correctness of the workflow process in presence of other concurrently executing workflows according to the algorithm provided in [B. Arpinar et al, 1999],[B. Arpinar, 1998].

Task Agent. Task agents act as wrappers of the actual applications. A typical task agent knows the meta-model of the task that it is associated with and the procedures for executing the task or accessing the database if it is a database task or collaborating with the users if the task is a human task. It also communicates with scheduling agents and recovery agents to report the current situation of the task (e.g., committed, failed, executing, etc.). There are several types of tasks [J. Miller et al., 1998]; transactional, non-transactional, user and web. Transactional tasks are those that support ACID (Atomicity, Consistency, Isolation and Durability) properties. Non-transactional tasks are used when an ordinary application that does not enforce atomicity or isolation is included in the workflow. A user task is used for purely manual tasks like a phone call of a user. Web tasks are those that involve a web application.

Depending on the type of the task they are in charge with, task agents have different capabilities. For example, for user tasks, task agents have worklist management capability.

Recovery Agent. Recovery agents have the knowledge of currently running and past instances of the workflows that they are associated with. They communicate with scheduling agents and task agents to gather information about running instances. A recovery agent has access to the history database and should have mechanisms to query this database. When a failure in a task or an unanticipated change in the specification occurs, recovery agent takes the initiative. It produces the new path or definition to follow by making backward recovery using a compensation mechanism. It determines which tasks to compensate and what to do next, and informs the relevant scheduling agent about the new path.

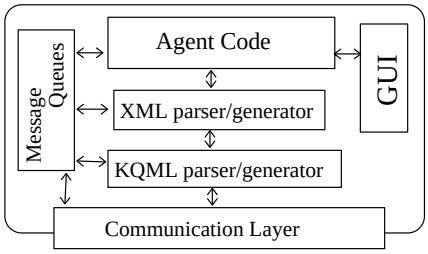


Figure 1. Structure of an Agent

Facilitator Agent. It acts as a facilitator (in KQML terminology) for agents in the system. It collects advertisements of the agents in terms of their capabilities and facilitates agents to find each other to satisfy their needs. In order to increase efficiency, there should be more than one facilitator in the system. Facilitator agents should know each other's address and query each other to answer requests of the agents. Therefore in our system, facilitator agents advertise themselves to other facilitator agents.

The general structure of an agent is shown in Figure 1. This structure is common to all agents in the system with only one exception that only interface agents have graphical user interfaces (GUIs). There is an XML parser/generator that helps agents to understand the content of KQML messages and forming a new KQML message containing some XML content to be sent to another agent. Message Queues are used to keep track of incoming and outgoing messages so that an agent knows which message to respond and which messages it has sent and has not receive a response yet. Agent code includes the procedures and modules that help agents in realizing their functionality.

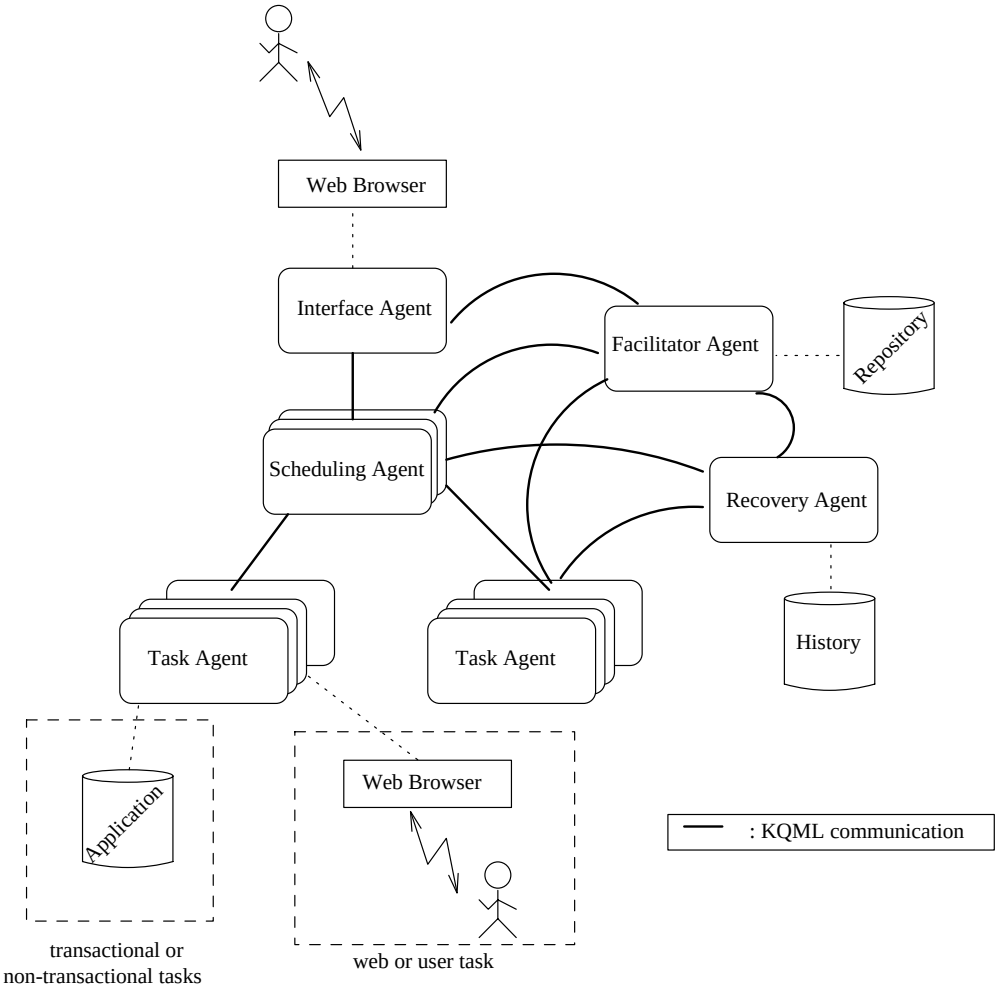


Figure 2. Workflow Management System Architecture

Figure 2 shows the overall architecture consisting of the components and agents taking part in our workflow management system. Workflow system is initiated by the user through a web browser. Interface agent constructs the definition in XML conforming to a *workflow.dtd* (see Section "Workflow Process Definition"). In order to start execution, interface agent sends this definition to a scheduling agent capable of interpreting the definition. The scheduling agent may enact the whole process or delegate some parts of it to other scheduling agents. In this way, an execution tree is formed having scheduling agents at interior nodes and task agents at the leaves. During enactment, a scheduling agent needs to find other scheduling and/or task agents capable of doing the required work. This is facilitated by the facilitator agent that holds a repository for storing agent advertisements.

Since agents are autonomous, there are no predefined control dependencies among them, therefore, if an agent requires a task which is managed by another agent, it cannot simply instruct it to start the task [N. R. Jennings et al., 1996]. Instead, agents should come to a mutual agreement about the terms and conditions under which the task is to be performed through negotiating with each other. We call this type of negotiation as *task-oriented negotiation*. We use an auction mechanism, Vickrey auction [W. Vickrey, 1961], for this type of negotiation. In Vickrey auction model, each of the participants of the auction submits a sealed bid known only by the receiving party. The participant with the lowest (highest) bid is awarded the work at the second lowest (highest) bid price. For task-oriented negotiation, our auction model supports more than one term in the bid. Therefore, we have made a slight modification to the Vickrey auction model. An agent should determine the winner agent not only by looking at price offerings but also other terms such as the proposed duration of the work. This is achieved by assigning percentages of importance to the terms involved in the negotiation. In this way, a single value is obtained and the rules of the auction are applied as before.

Thus, in agent interactions described above, all agents use *task-oriented negotiation* during work delegation and task invocation.

Workflow Process Definition

There is a need to define the work to be accomplished and also the ordering principles (i.e., control-flow) among the activities of workflow processes. In our framework, we choose to use XML to define a workflow process. In this way, a workflow definition can be interpreted by all other agents or components that have the capability to parse an XML document and have access to the DTD used.

XML by itself is not enough to enable plug and play workflows. XML makes it easy to create specialized markup languages that identify and describe workflows, the goods or services, and the numerous other document types. But if every business uses its own XML definitions for describing its workflow it is not possible to achieve interoperability. In this respect CBL will include a workflow and service description in its future versions [VEO Systems Inc., 1998]. For the

time being we have developed our own DTD for workflow description to be replaced by the original when it becomes available.

In our approach, a workflow process is defined as a collection of blocks, tasks and subprocesses [A. Dogac et al., 1998a]. Processes and tasks have input and output parameters. There are eight block types in our workflow definition; serial, and-parallel, or-parallel, xor-parallel, conditional, for-each, iterative and contingency. Activities (i.e., tasks, blocks or subprocesses) in a serial block should be performed one after another. Similarly, activities in a parallel block are executed in parallel. And/or/xor parallel blocks differ in the sense that termination of all/some/one of the activities is required for the termination of the block. Conditional block allows to constrain the execution of activities according to the result of a conditional expression. For-each block is necessary when the same activities are to be performed for each member of a list. An activity in an iterative block is repeated as many times as the iteration parameter requires. A contingency block is used to define alternative paths in the definition. In an agent-based workflow system, external events should also be handled since some activities are activated according to messages coming from other agents. Therefore, external variables are also used in workflow definition and activities are placed in conditional blocks that are controlled by conditions on external variables. The values of external variables are dynamically updated and the flow of workflow process is determined accordingly at run-time.

A document type definition, namely *workflow.dtd* used to describe a workflow process having the above features is provided in Appendix A.

In the following, we provide an example workflow process written in XML conforming to the *workflow.dtd*.

Example 2.1

The following is a simplified workflow definition of the selling process of a bookstore written in XML. Parameters of some tasks and expressions for conditions (see dots in the definition) are omitted to save space. The process is also shown graphically in Figure 3.

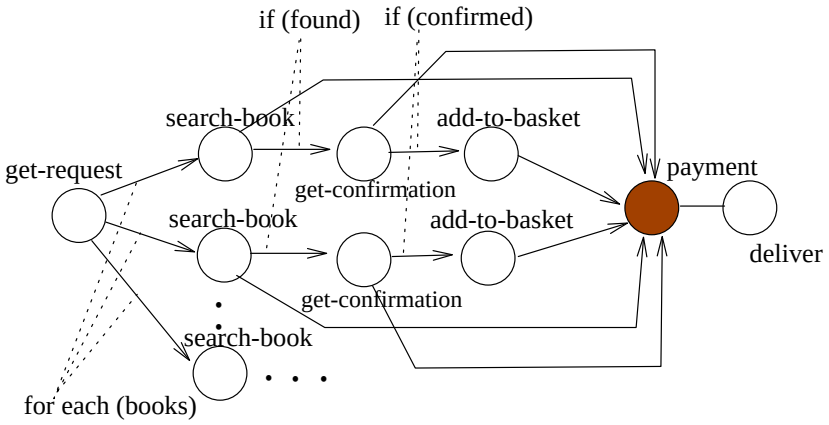


Figure 3. The Selling Process in a Bookstore

```
<process name='bookstore'>
  <variables>
    <var type=INT>customer-id</var>
    <var type=LIST> <list type=XML>books</list></var>
    <var type=INT>found</var>
    <var type=INT>confirmed</var>
    <var type=LIST> <list type=STRING>basket</list></var>
  </variables>
  <block>
    <task name='get-request' description='http://www.srdc.metu.edu.tr/mpd/get-
request.RDF'>
      <parameter mode=IN>customer-id</parameter>
      <parameter mode=OUT><list>books</list></parameter>
    </task>
    <for-each-block type=AND-PARALLEL>
      <list>books</list>
        <task name='search-book' description=' http://www.srdc.metu.edu.tr/mpd/search-
book.RDF'>
          <parameter mode=IN><list-element>books<index><int>i</int></index>
                    </list-element> </parameter>
          <parameter mode=OUT>found</parameter>
        </task>
        <conditional-block>
          <condition> ...  </condition>
            <task name='get-confirmation' ...> ... </task>
        </conditional-block>
```

```
        <conditional-block>
          <condition> ...  </condition>
          <task name='add-to-basket' ...> ... </task>
        </conditional-block>
    </for-each-block>
    <task name='payment' type=SUBPROCESS ...> ... </task>
    <task name='deliver' role='postman'
description='http://www.srdc.metu.edu.tr/mpd/deliver.RDF'> ... </task>
  </block>
</process>
```

The customer request is obtained as a list of book descriptions (*books*). For each of the books in the list, search from the bookstore database is done in parallel. Customer confirmation is necessary for the books that are found in the bookstore. Confirmed books are added to the basket of the customer (*basket*). At the end, payment subprocess is executed and delivery of the books is scheduled. If there is nothing in the basket, payment and delivery processes are bypassed. Payment subprocess may include many tasks one of which may be to get the credit card number of the customer. Delivery may be a user task to be scheduled to a person as a work item who satisfies the role given in the definition (i.e., postman) or it can be a subprocess to find out the best possible delivery service for the specific user and the item.

Task and Role Definitions

The workflow definition involves many types of tasks such as transactional tasks, user tasks, etc. as described earlier. In order for agents to advertise themselves as capable of executing a task or to accept a task execution offer, there is a need for a mechanism to describe the metadata of tasks. Resource Description Framework (RDF) [RDF, 1998] is used in this respect to define the metadata of individual tasks.

RDF is a foundation for processing metadata for providing interoperability between applications that exchange machine-understandable information. The model of RDF is represented by named properties and property values. The basic data model consists of three object types: *resources* which are the things being

described by RDF, *properties* which are specific aspects, attributes or relations describing a resource and *statements* that assign a value to a property of a resource. For example the sentence "Sena is the creator of "http://www.srdc.metu.edu.tr/~nural" is a statement in RDF, the resource is "http://www.srdc.metu.edu.tr/~nural" and the property is "creator". The RDF data model provides an abstract, conceptual framework for defining and using metadata. A concrete syntax is also required and XML is used for this purpose in [RDF, 1998]. Meaning in RDF is expressed through a reference to a schema that defines the terms that will be used in RDF statements and gives specific meanings to them. A variety of schema forms can be used with RDF one of which is defined in [RDFSchema, 1998] that has some specific characteristics to help with automating tasks using RDF. There are also some containers defined in RDF; Bag, Sequence and Alternative to refer to a collection of resources.

The RDF metadata descriptions are stored in the marketplace directory to be used by the agents. Pointers to the descriptions are given in workflow process definition as an attribute as shown in Example 2.1.

Below is an example metadata description of a task, namely 'deliver' given in Example 2.1.

Example 2.2

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
  xmlns:t=" http://www.srdc.metu.edu.tr/mpd/schemas/task#">
  <rdf:Description about="http://www.a.b.c/deliver.xml">
    <t:name>deliver</t:name>
    <t:type>UserTask</t:type>
    <t:duration>12 hours</t:duration>
    <t:priority>1</t:priority>
    <t:deadline>25/11/98</t:deadline>
    <t:participant>
     <rdf:ALT>
       <rdf:li> delivery-company </rdf:li>
       <rdf:li> postman </rdf:li>
```

```
        <rdf:li> office-boy </rdf:li>
      </rdf:ALT>
    </t:participant>
  </rdf:Description>
</rdf:RDF>
```

In the above RDF description, the schema for task description, namely "http://www.srdc.metu.edu.tr/mpd/schemas/task" is defined elsewhere (resides in the marketplace directory) by using RDF Schema specification described in [RDFSchema, 1998].

Furthermore, in order to assign user tasks of a workflow to a user capable of performing them, we need metadata of the users and roles involved in the workflow management system. These role and user descriptions are also written in RDF. An example role definition is provided below.

Example 2.3

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
  xmlns:r="http://www.srdc.metu.edu.tr/mpd/schemas/roles#">
  <rdf:Description about="http://www.srdc.metu.edu.tr/roles/postman#">
    <r:users>
      <rdf:BAG>
        <rdf:li resource="http://www.a.b.c/users/John.Doe/>
        <rdf:li resource="http://www.a.b.c/users/Mary.Doe/>
        <rdf:li resource="http://www.a.b.c/users/Bob.Smith/>
      </rdf:BAG>
    </r:users>
  </rdf:Description>
</rdf:RDF>
```

Communication among Agents

In our framework, we use KQML ([Y. Labrou and T. Finin, 1994],[Y. Labrou and T. Finin, 1997]) for inter-agent communication. KQML is a language and a protocol for exchanging information and knowledge.

KQML uses a set of standard message types an example of which is given below:

```
(ask-one
  :content (PRICE IBM ?price)
  :receiver stock-server
  :language LPROLOG
  :ontology myOntology )
```

In KQML terminology, "ask-one" is a performative. A performative sets parameters that are introduced by keywords such as "sender", "language", etc. Above message represents a query about the price of a share of IBM stock. Ontology assumed by the query is identified by the token "myOntology", the receiver of the message is "stock-server" and language used is "LPROLOG".

The performatives of KQML are organized in three categories. Discourse performatives (*ask-if, ask-all, ask-one, stream-all, tell, insert, delete-one, delete-all, achieve, advertise, subscribe,* and some more) are used in the context of an information and knowledge exchange kind of discourse between two agents. Intervention and mechanics of conversation performatives are used either prematurely terminate a conversation (*error, sorry*) or override the default protocol (*standby, ready, next, rest* and *discard*). Networking and facilitation performatives allow agents to find other agents that can process their queries (*register, forward, broadcast, broker-one, broker-all, recommend-one, recommend-all, recruit-one, recruit-all,* and some more).

KQML is essential so that the agents of heterogeneous nature can communicate. However, since KQML is designed for knowledge sharing, certain aspects of the language are inadequate for agent negotiation. If negotiation is implemented using existing performatives of KQML, it becomes necessary to associate a new interpretation to the performatives, which has not been originally intended for. As an example, when we consider a price negotiation between two agents, the capabilities offered by KQML in this respect could be the use of *insert* performative. That is, an agent, say A_1, could insert the price it is offering to the counter agent's (A_2) knowledge base (KB) and A_2 could see whether this is acceptable. If not, A_2 might make a counter proposal by inserting a new price to

the KB of A_1. However, this is not a natural way of handling negotiation. First, an agent must have been advertised that it will accept such an insert. Secondly, what is inserted is not a global fact but only the items of negotiation. For the above reasons, we propose to add the following three performatives to KQML: *propose* and *counter-propose* and *accept*. In the following, their syntax and intended meanings are provided:

(propose
:sender <word>
:receiver <word>
:in-reply-to <word>
:reply-with <word>
:language <word>
:ontology <word>
:content <conditions>)

This performative indicates that the :content expression of the message whose id is specified :in-reply-to is true of the :sender under the conditions specified in :content. This performative should be sent :in-reply-to one of *ask-all, ask-if, ask-one, stream-all* or *achieve* messages. It is intended for providing the terms of negotiation.

(counter-propose
:sender <word>
:receiver <word>
:in-reply-to <word>
:reply-with <word>
:language <word>
:ontology <word>
:content <conditions>)

This performative is sent :in-reply-to a previous *propose* message. It is intended for providing a counter proposal to a previous message.

(accept
:sender <word>

:receiver <word>

:in-reply-to <word>

:reply-with <word>

:content <condition>)

This performative is used to terminate the negotiation with agreement under the conditions referred in :content. It must be sent in-reply-to a previous *propose* or *counter-propose* message. *Sorry* performative of KQML can be used also to terminate the negotiation, but this time without an agreement.

Having defined these three new performatives, below we provide an example in which agents negotiate via these performatives.

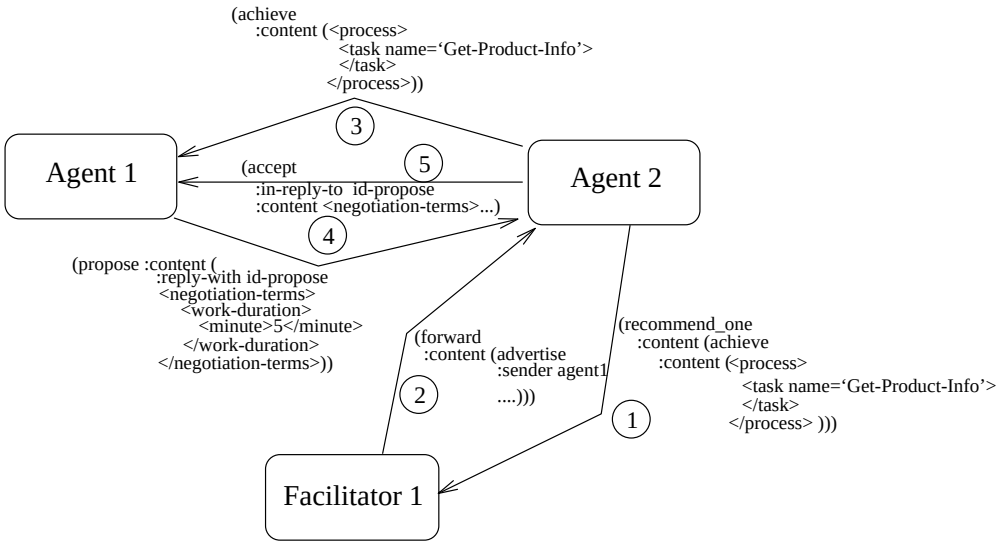


Figure 4. Task-oriented Negotiation Example

Example 2.4

In Figure 4, a task-oriented negotiation between two agents is illustrated. Agent 2 wants to have the task "Get-Product-Info" performed by another agent that is capable. Agent 1 has been advertised to the facilitator agent that it can accept *achieve* messages containing this task as the content, that is, Agent 1 is capable of executing (achieving) "Get-Product-Info" task. Normally, there might be more than one agent, which has advertised but we show only one agent (Agent 1) in this example. First, Agent 2 contacts with the facilitator agent and asks for an agent

that is capable of performing the task (1). Secondly, facilitator recommends Agent 1 to Agent 2 for the task in question (2). Then an auction (task-oriented negotiation) is started by Agent 2 through sending the 3rd message. Agent 1 replies this message with a *propose* message (4) and if Agent 2 has not received any better proposal, it accepts the offer of Agent 1 and confirms this by sending the *accept* (5) message together with the conditions accepted in the content part. After this point, it is the responsibility of Agent 1 to execute the task and to send any results back to Agent 2. Only the important parts of the messages are shown in the figure.

Marketplace Architecture

Having explained our workflow management system approach, we now discuss how this approach is used in building an electronic marketplace, MOPPET where sellers and buyers meet and negotiate to make the best deal.

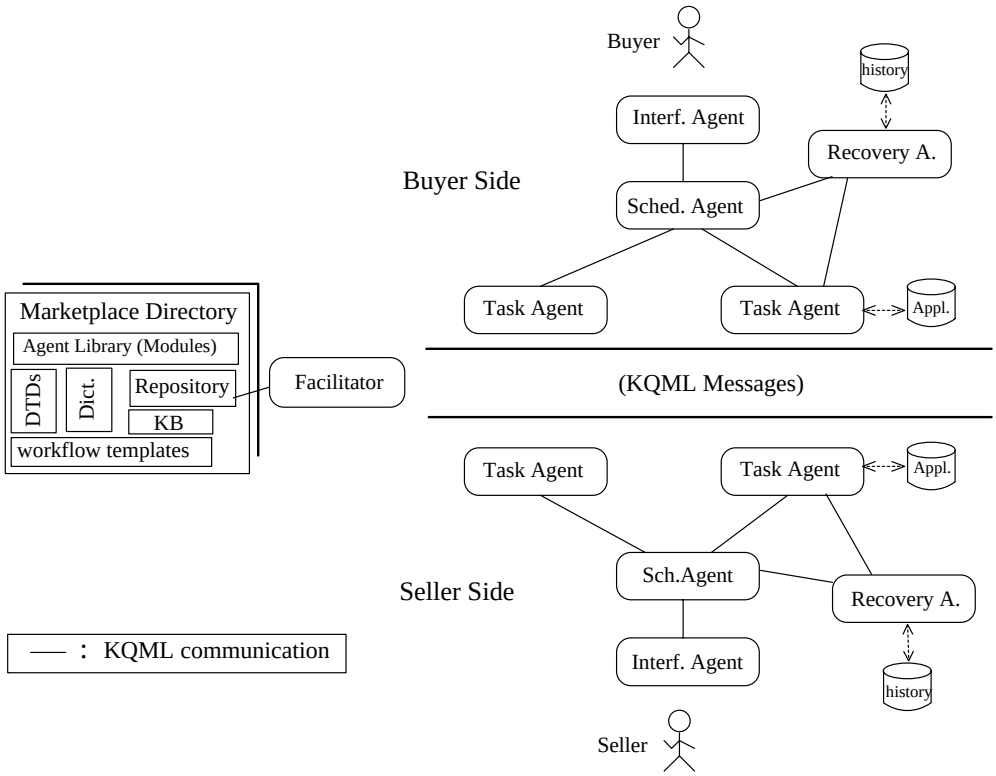


Figure 5. Architecture of the Marketplace, MOPPET

As discussed before, there are various electronic commerce models such as e-shops, e-procurement models, e-malls, value chain integrators, third party

marketplaces etc, as classified in [P. Timmers, 1998]. In this paper, we describe an electronic marketplace model using the agent-based workflow management system presented in previous sections. Note that, however, other models can also be realized using the workflow management system as the basis and adding (or removing) a few special components to meet the specific needs of other models.

Electronic commerce, which is a complex business process itself, cannot be modeled effectively by the current marketplaces that support buyer/seller behavior in an overly simplistic manner. For this reason, we propose to exploit workflow systems to model buying and/or selling processes. In other words, instead of a single buying/selling agent, a number of agents are coordinated to realize the electronic commerce processes.

At the core of MOPPET (Figure 5), there is a directory which provides document type definitions (DTDs), a dictionary of synonyms, repositories to be used by the facilitator agent, workflow templates, a knowledge base (KB) for item compatibility checks and some library modules for agents' own use.

In the following, the use of main components in the marketplace directory is explained.

Document Type Definitions (DTDs) - We use DTDs in various parts of the system for different purposes such as:

- to describe the workflow process (*workflow.dtd*)
- to describe the customer interests (*customer.dtd*)
- to describe the individual items/services (*computer.dtd, car.dtd,* etc.)
- to describe the terms of negotiation (*terms.dtd*)

We have already mentioned about *workflow.dtd* in Section "Worklfow Process Definition". The *customer.dtd* is used by buyers to describe themselves and their interests so that a seller who is trying to sell a product can find those buyers who might be interested in. The following is the part of *customer.dtd* that we are currently using:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE simple [
<!ELEMENT customer (address?,interested-in)>
<!ATTLIST customer
     customer-id     ID    #REQUIRED>
<!-- address element description is omitted -->
<!ELEMENT interested-in (item+)>
<!ELEMENT item ANY>
\]>
```

The DTDs mentioned in the third item are detailed product descriptions that are to be produced by industry groups. In this respect, there is an effort by RosettaNet [RosettaNet, 1998] to produce specifications for computers that can possibly be implemented through XML DTDs. In fact, Common Business Library (CBL) [VEO Systems Inc., 1998] includes a DTD for laptop computers, namely *laptop.dtd* which conforms to the specification generated by RosettaNet [RosettaNet, 1998]. Throughout the paper, we are using our own DTDs to be replaced by the originals when they become available.

During negotiation, agents must use the same vocabulary for terms of negotiation. For example, if an agent uses hours for duration and the other one uses minutes, they cannot be compared with each other if it is not known whether the scale is minutes or hours. In order to achieve a common terminology for the terms of negotiation, we use *terms.dtd* given in the following. In producing this DTD, we use the modules provided by CBL. CBL includes a group of DTDs for weights and measures, the description of time and location and the terms of trade. For example, "transaction.unit.of.measure" is defined in *measures.mod* in CBL and "amount.monetary" is defined in *value.mod*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE simple [
<!ELEMENT negotiation-terms (price?,work-duration?,quality?,quantity?,delivery-time?)>
<!ELEMENT price (amount.monetary)>
<!ELEMENT work-duration (year?,month?,week?,day?,hour?,minute?,second?)>
<!ELEMENT quality ANY>
```

<!ELEMENT quantity (transaction.unit.of.measure)>
<!ELEMENT delivery-time (hour?,day,month,year)>
/]>

Agents give the names of DTDs in the :ontology part of the KQML messages so that they can understand the context of each other's questions or responses as demonstrated in Example 3.2.

Dictionary of Synonyms - Interface agents are responsible for obtaining the product/service requests from the buyer. However, the user may not know the right term (i.e., the term used in DTD for that item) to use for the item s/he is looking for. Therefore, a dictionary of synonyms which evolves over time should be used [A. Dogac et al., 1998b]. For example, consider a computer store that uses a *computer.dtd* in describing its products and services. If a buyer wants to buy a "CPU" and uses the "Processor" while "CPU" is the term used in the DTD, then the dictionary of synonyms is searched to match the word "Processor" with "CPU".

The dictionary of synonyms is empty at the very beginning. During marketplace activities, however, it learns the synonyms for the terms with the help of users and afterwards and it can answer queries about the terms it has learned so far.

Knowledge Base for Compatibility Requirements - There should be two sets of rules in the knowledge base.

- relationships among items. For example, a customer who buys a computer can be interested in purchasing a printer. These rules are checked by the interface agent in order to be able to provide related offerings to the buyer.
- compatibility requirements among items. For example, a customer may want a software product that has a specific memory requirement. These compatibility rules are checked when a buyer makes a request so that if there are incompatibilities among the items requested, the buyer is warned and the request is adapted.

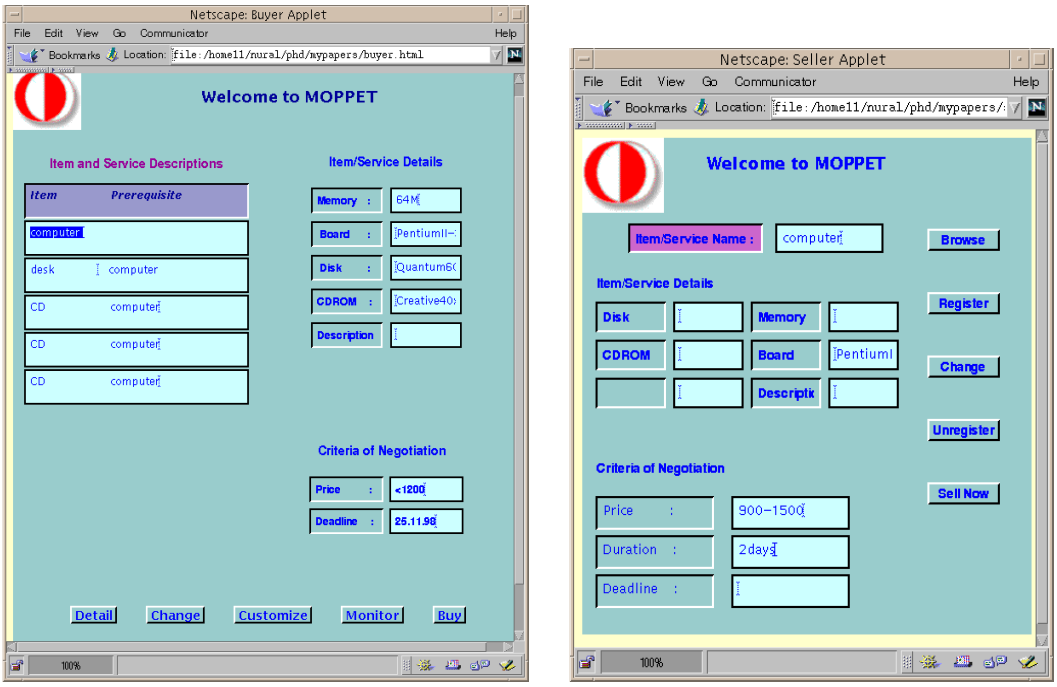
Currently, we are developing a simple knowledge base to contain above information. However, there is a need for further research and development for representing and processing these relationships among items.

Agents' Responsibilities in the Marketplace

The general functionality of agents is described in Section "Agent-based WfMS Architecture". In this section, the features and the responsibilities of the agents specific to the marketplace environment are described.

Interface Agents - We differentiate two main types of users: buyers and sellers and two modes of these users: active and passive. Active buyer is the buyer who makes requests for items to be bought and passive one waits until a request comes from a seller who is trying to sell his/her product. Similarly, an active seller tries to sell a specific product and the passive one waits for buyers who are interested in buying his/her products. These roles are introduced for modeling different marketing policies. Interface agents, especially the graphical user interfaces of the agents, differ for these types and modes of users. In the following, we briefly describe the functions of interface agents for both modes of sellers and buyers:

- For an active buyer: Interface agent of an active buyer is responsible for constructing the initial workflow process definition. The buyer gives the items and/or services s/he wants to purchase and also dependencies among the items if there are any, through a Java applet provided by the interface agent as shown in Figure 6a. The buyer may not know the details of the item/service s/he wants. For example, s/he may want to buy a computer but s/he may not know the properties of the computer such as CPU, board, etc. This information is searched by the interface agent from the DTDs in the marketplace directory with the help of a particular task agent (capable of doing such a search). Then, the fields obtained from the DTD are presented to the buyer (see Item/Service Details part in Figure 6a) and filled by the buyer if there are preferences for them. The buyer can also provide negotiation details for the items s/he requested if there are any.



(a) (b)

Figure 6. Interfaces for (a) Buyers and (b) Sellers

Additionally, interface agents are capable of providing related items or services once a buyer requests an item. For example, a customer looking for a computer can also be interested in buying some software suitable for his/her computer. These related items can automatically be offered to the customer by the interface agent. To achieve this functionality, a knowledge base that holds the rules and compatibility requirements among the items is searched. This knowledge base is stored in marketplace directory as described before.

Interface agent of the buyer translates the buyer inquiry to the workflow definition in XML by modifying the appropriate workflow template obtained from the marketplace directory and finds a scheduling agent by starting an auction among suitable scheduling agents. Then it transfers the work to the winner scheduling agent through a KQML message. The workflow templates for each type of buyers and sellers will be described in the next section.

In addition, throughout the workflow process execution, if user intervention is required the interface agent handles it. Monitoring of the workflow can be

added to the capabilities of interface agent by providing a module (from agent library) to the agent.

- For a passive buyer: Interface agent of the passive buyer is activated when a buyer registers (advertises) himself to the marketplace by providing his interests (conforming to *customer.dtd*). An example advertisement is shown in Example 3.2. Afterwards, it waits idle until a request comes from a seller. From this point on, interface agent adapts the workflow template and activates a workflow instance by finding a suitable scheduling agent through *task-oriented negotiation* as described before.

- For a passive seller: Interface agent for this type of seller is mainly responsible for registering the seller to the marketplace. It can do this by listing the previously registered catalogs such as "computer", "book", etc. and seller chooses the suitable one for him/her. Or the catalog to be registered can be a new one (Figure 6b). Sellers who have more than one item/service should register for each of the items/services.

  The interface agent uses *advertise* KQML message to introduce the seller to the marketplace. An example advertisement message for a seller is provided in Example 3.2. The registered catalog name must exist in one of the DTDs that the marketplace knows or has access to.

  The interface agent of the passive seller also obtains a workflow template from the marketplace directory. The workflow is initiated by a request coming from a buyer similar to the case of passive buyer.

- For an active seller: The seller might also initiate a selling process by providing the details of the product s/he is trying to sell. This type of interface agent provides a GUI similar to the one provided by the interface agent of the buyer. Whenever the seller clicks the "Sell" button, the interface agent initiates a workflow process by contacting a scheduling agent and sending it the workflow definition obtained by adapting the template for active sellers (Figure 6b).

Apart from sellers and buyers, there might be interface agents for other end-users of the workflow system. For example, a person who is responsible for physically delivering a product to a buyer has an interface agent. This type of interface agents has a different type of GUI that displays the work items assigned to the users. In other words, some interface agents are only responsible for providing the worklists to the users.

The interface agent types described above are the ones required for realizing a electronic marketplace environment. If other electronic commerce models such as supply-chain integration, e-procurement, e-mall etc ([P. Timmers, 1998]) are to be realized, different interface agents must be implemented. The underlying workflow system can be used for any kind of workflow processes that such models require.

Task Agents - The marketplace contains many types of specific task agents in addition to the task agents of other workflow activities. These can be categorized as:

- Negotiation task agents: These are used for negotiation between sellers and buyers for a specific item. We name this type of negotiation as *item-based negotiation*. For this type of negotiation, task agents uses negotiation criteria of the users obtained at the beginning by the interface agents. A number of *propose* and *counter-propose* messages are exchanged between the task agents of the seller and buyer sides until an agreement is reached. Negotiation task agents should know what to offer next and how to determine the end of negotiation. Many attributes may be negotiated between the sides such as price, deadline, delivery time, etc. Therefore, a multi-issue, multi-party negotiation algorithm is used for item-based negotiation [M. Yilmaz, 1999].

  Negotiation model is based on Raiffa's bilateral (two parties, many issue) negotiation [H. Raiffa, 1982]. This bilateral negotiation model is transformed into multilateral (many parties, many issues) negotiation model by using a set of mutually influencing two parties, many issues negotiations [P. Faratin et al., 1998]. This type of negotiation is also called as integrative bargaining.

Sequence of offers and counter-offers in two-party negotiation is called as a negotiation thread. Offers and counter-offers are generated by linear combinations of simple functions, called tactics. The term strategy is used to denote the way in which an agent changes the weights of the different tactics over time. By the help of strategies agents make use of internal reasoning model and history of negotiations and could influence other negotiation threads.

In the bilateral negotiation model, agent scoring function stemmed from MAUT is used in order to decide which offer is better than other, each agent has a scoring function which returns the score of a given issue, that is, it assigns a value to an issue in the range of acceptable values and these scores are kept in the interval [0,1] for convenience. The next element of model is relative importance that an agent assigns to each issue under negotiation. These weights are normalized, that is sum of weights of each issue is 1. With these elements, an agent scoring function for a contract is defined as sum of scores of each issue multiplied by its weight.

In order to prepare counter offer that is next offer in the negotiation thread agent uses a set of tactics that generate new values for each variable in the negotiation set. Three different families of tactics based on the needs of business process applications are developed [P. Faratin et al., 1998]; time-dependent, resource-dependent and behaviour-dependent.

Tactics are set of functions that compute the value of an issue by considering a single criterion. The set of values for the issues are then the range of the function, and the single criteria -time, resource, ...- is its domain. Agents may want to consider more than one criterion to compute the value for single issue; this could be achieved by generating counter proposals as a weighted combination of different tactics covering the set of criteria.

In order to determine the best course of action, which will result in an agreement on a contract that maximizes the scoring function, agent utilizes the

strategy. When an agent receives an offer that is unsatisfactory, it generates a counter offer and it uses different combinations of tactics for a particular issue.

The further details of the negotiation can be found in [M. Yilmaz, 1999].

- User task agents: When a user activity is to be executed by the scheduling agent, this type of task agent is required. User task agent stores the request (work item) into a worklist. User task agent should decide on which user this work is to be scheduled. It achieves this by looking at RDF metadata of rules and users that exist in the marketplace directory as explained in Section "Task and Role Definitions". According to the decision, it sends appropriate KQML message to the interface agent of the user. For example, suppose the work to be assigned is the delivery of a product to a buyer. Suppose also that this is a manual task and there are five people who can deliver this package. The user task agent first finds the interface agents of those five people through the RDF metadata of 'postman' role (see Example 2.3) and negotiates with them to find the most appropriate one. Then, this work item is stored into the worklist of that person and interface agent for that person is informed about this new work item.

- Query task agents: These task agents are required for querying especially the seller catalogs. Sellers export their catalogs as XML pages and throughout the marketplace activities, these catalogs need to be searched for specific items/services. We use XML-QL that is a submission to World Wide Web Consortium (W3C) [A. Deutsch et al., 1998] for querying XML documents. There are many efforts going on querying semi-structured data ([S. Abiteboul et al., 1997],[P. Buneman et al., 1996]) and SGML [K. Bohm et al., 1997] and some of the approaches are adapted also to query XML documents [D. Suciu, 1998]. Query task agents uses Document Object Model (DOM) [DOM, 1998] implementation together with an XML parser and a XML-QL query processor to answer queries. DOM provides a uniform mechanism to access and manipulate parsed XML or HTML documents. Specifically, DOM defines an object-oriented API of an XML document.

Example 3.1 shows an example query written in XML-QL for querying a computer shop's catalog.

Example 3.1

The computer shop's catalog contains product entries conforming to the following simplified DTD:

```
<!ELEMENT item (computer,description?,price_info,item*)>
<!ELEMENT computer (memory,board,cdrom?,disk)>
<!ELEMENT description (paragraph | img)* >
<!ELEMENT img EMPTY>
<!ATTLIST img src CDATA #REQUIRED>
<!ELEMENT paragraph (#PCDATA)>
<!ELEMENT price_info (productno, price, avail, warranty?)>
```

The following is a query written in XML-QL that asks for the computers having disks 4GB or larger and having price lower than 1200$.

```
WHERE <item>
        <computer> <disk> $d </> </>
        <description> <price_info> <price> $p </> </> </>
      </> IN "www.a.b.c/computer.xml",
      d > 4GB, p < 1200
```

Facilitator Agent - Apart from facilitating the agents of the workflow system, facilitator agent is also responsible for storing advertisements of the sellers and the buyers. In fact, these advertisements are also done by the agents, but they serve as the catalog information, which is specific to a marketplace environment. In the following example, several advertisement messages that should be sent to the facilitator agent are described.

Example 3.2

The following is an *advertise* message of a seller (seller1) who sells computers:

```
(advertise
```

```
    :sender    seller1
    :receiver  facilitator1
    :language  KQML
    :ontology  kqml-ontology
    :content   (ask-all
                :receiver    seller1
                :language    XML
                :ontology    xml-ql
                :content     ( WHERE <computer>$c</>
                               IN "http://www.a.b.c/seller1.xml")))
```

This message says that the agent seller1 can respond to *ask-all* messages that contains a query about a "computer" in seller's catalog (seller1.xml).

Similarly, a buyer (buyer1) who is interested in purchasing a computer can use the following message:

```
(advertise
    :sender    buyer1
    :receiver  facilitator1
    :language  KQML
    :ontology  kqml-ontology
    :content   (ask-one
                :receiver   buyer1
                :language   XML
                :ontology   customer.dtd
                :content      (<?xml namespace name=" http://www.srdc.metu.edu.tr/mpd/
computer.dtd" as "c" ?>
                              <customer customer-id='1234'>
                                <interested-in>
                                  <item> <c:computer> </c:computer>
                                  </item>
                                </interested-in>
                              </customer> )))
```

Other agents in the system should also advertise themselves by giving the capabilities offered. A sample message from a scheduling agent (sch1) is the following:

```
(advertise
    :sender     sch1
    :receiver   facilitator1
    :language   KQML
    :ontology   kqml-ontology
    :content    (achieve
                :receiver    sch1
                :language    XML
                :ontology    http://www.srdc.metu.edu.tr/mpd/workflow.dtd
                :content     (<process> </process> )))
```

This message indicates that sch1 can execute the given workflow definition in *achieve*$_{:content}$. Since the definition includes only the <process> tags, not a specific process or a task it means that it can schedule all kinds of workflow processes conforming to *workflow.dtd*.

Templates for Seller and Buyer Workflows

In the marketplace, there are mainly two types of parties: buyers and sellers. We further categorize these parties as passive and active buyers/sellers. A buyer is active when s/he wants to buy items and/or services and it is passive when s/he prefers to wait for some seller to come and try to sell him/her items and/or services. The same holds for the sellers as well.

The templates reside at the marketplace directory and are configured by the interface agent according to the mode (active or passive) of the user. A buyer or a seller can behave both as active and passive at the same time by contacting more than one interface agent. In the following, the simplified versions of the templates generated by the interface agent for each mode of both buyers and sellers are provided (XML versions of the templates are given in Appendix B).

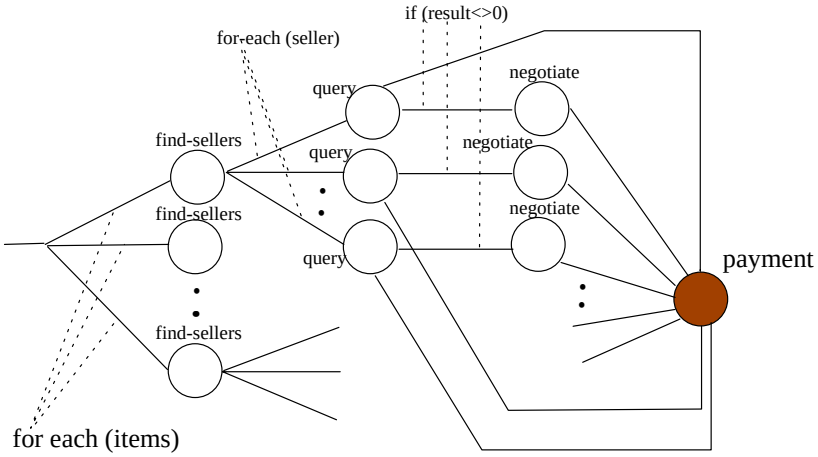


Figure 7. Workflow Template for Active Buyers

Workflow Template for Active Buyer: The template given in Figure 7 is further modified when a buyer inquiry is made. For example, "for-each (items)" is replaced with a series of other types of blocks when the constraints and the orderings among the items are collected. Note that, if there is no dependency or a specific order among the items, "for-each (items)" block is preserved and only the details of items are included in the definition. These modifications will be clear with the example given in the following section.

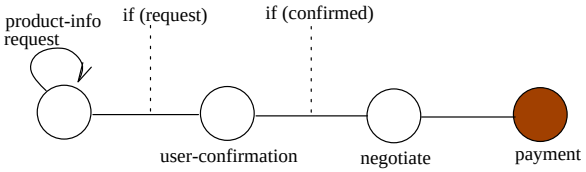


Figure 8. Workflow Template for Passive Buyers

Workflow Template for Passive Buyer: Workflow of the passive buyer (Figure 8) is initiated by external events/variables, which comes from a seller workflow. 'User-confirmation' is the task to be done when a 'request' and 'product-info' comes from an active seller. After this point, the flow is the same with the active one. 'Negotiate' task is scheduled and according to the result of negotiation, 'payment' subprocess is to be scheduled.

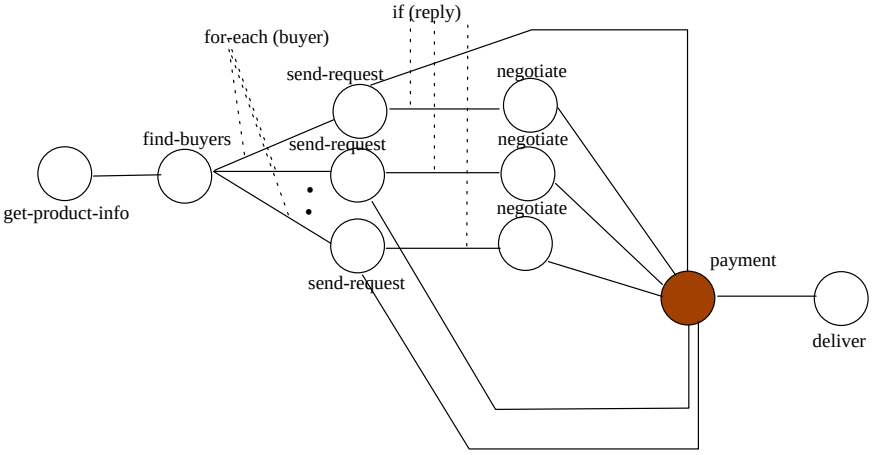


Figure 9. Workflow Template for Active Sellers

Workflow Template for Active Seller: Workflow template of active sellers (Figure 9) is similar to the one for active buyers. Product details are obtained from the seller through the interface and then 'find-buyers' task is scheduled to find possibly interested buyers. Finding interested buyers is done by the facilitator agent by looking at advertisement messages of the buyers conforming to the *customer.dtd* (see Example 3.2). To each of the buyers found, a request is sent. For those who replied positively, 'negotiate' tasks are scheduled. Finally, according to the result of negotiations, payment and delivery are scheduled.

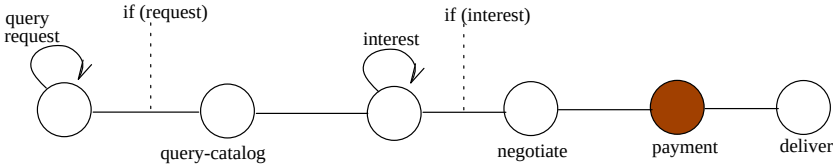


Figure 10. Workflow Template for Passive Sellers

Workflow Template for Passive Seller: The passive seller is activated by the message coming from an agent of the active buyer. This message initiates the workflow (Figure 10) and the first task is to query the seller's catalog. It sends the result of the query and waits to see if the buyer is still interested in buying the product. If so, as in other templates, 'negotiate' task is scheduled and 'payment' and 'deliver' tasks may also be invoked.

The Scenario

In this section, we provide a scenario for an active buyer, passive seller pair to further describe the MOPPET architecture.

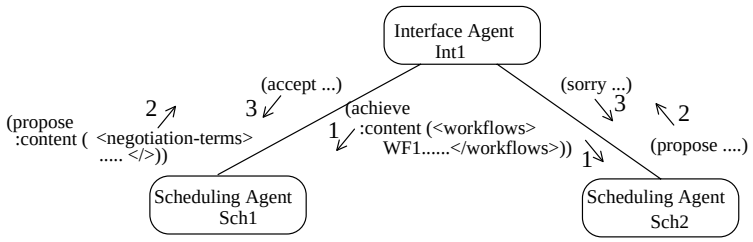


Figure 11. Finding a Scheduling Agent

When a buyer wants to buy an item from the marketplace it reaches the marketplace through its URL. The marketplace can be spread over many sites to provide scalability and availability. When more than one marketplace directory exist, they are linked together similar to the traders linked in Object Management Group (OMG)'s Trading Object Service specification [TOS, 1997]. Therefore, querying a marketplace directory involves querying the others that are linked to the current one if the current one can not answer the query.

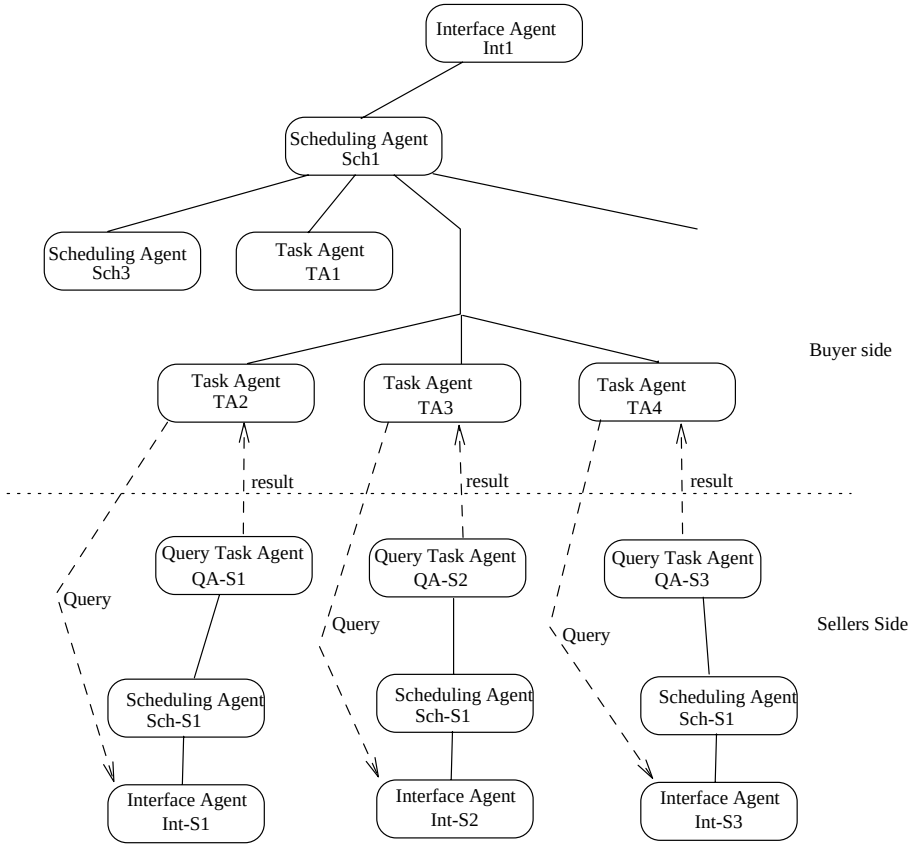


Figure 12. Interactions of Task Agents of the Buyer and Query Task Agents of Sellers

An interface agent is activated and this agent sends an applet to the buyer site. The buyer needs only a Web browser capability to conduct business at the marketplace. From this point on, the interface agent interacts with the buyer as shown in Figure 6a. According to the information gathered in this step, the interface agent adapts the related workflow template. This involves adapting the control flow and data flow in the template as well as adding or deleting activities or subprocesses. A user-defined subprocess can also be scheduled if the user wishes so, as long as the definition conforms to the *workflow.dtd*. The interface agent then looks for scheduling agents to schedule this workflow instance (Figure 11). A negotiation (*task-oriented negotiation*) with different scheduler agents are necessary for two reasons: (1) since schedulers are agents, they cannot be instructed to do things and (2) more importantly, some of these schedulers may not be capable of executing the workflow for several reasons like being already overloaded or not having access to some of the related task agents. An example *task-oriented negotiation* is illustrated in Figure 11. Interface agent of the buyer (Int_1) negotiates with two scheduling agents (Sch_1 and Sch_2) whose names are

recommended by the facilitator agent. It starts negotiation by sending a message containing an *achieve* performative. Then scheduling agents make proposals conforming to the *terms.dtd*. Among the agents, suppose that Sch_1 makes the best offer and therefore Int_1 sends it an *accept* message and sends Sch_2 a *sorry* message to finalize the negotiation.

Similarly, scheduling agent (Sch_1) which gets the job of enactment finds task agents to execute related tasks.

The workflow at the seller side is initiated when a buyer task agent sends a query to a seller's interface agent. The query is the event that activates the passive seller's interface agent (see Figure 10). The interface agent after adapting the template for passive sellers, finds a scheduling agent for that workflow instance. The scheduling agent starts executing the workflow instance and schedules the first task ("query-catalog") to a query task agent. We assume that the seller catalogs are expressed in XML and the query is expressed in XML-QL as mentioned previously. The results are sent to the buyer task agent who had originally sent the query. Once the buyers and sellers are thus identified the negotiation starts. This phase is shown in Figure 12.

Item-based negotiation is performed by negotiation task agents at both sides (Figure 13). Negotiation process at the seller side is initiated by an external "interest" event sent by a negotiation task agent of the buyer. The negotiation process is realized according to a multi-party, multi-attribute negotiation scheme as described in [P. Faratin et al., 1998] through the suggested *propose, counter-propose* KQML messages and finalized with a *sorry* or an *accept* KQML message. Once an agreement is reached, payment processes are scheduled at both sides. The task agent responsible for doing the payment at the buyer side gets the user preference for payment type such as credit card, on-site, etc. and other related information such as a credit card number. For user interaction, the agent contacts with the interface agent of the buyer. The task agent then sends this information to the seller side task agent that activates the payment process.

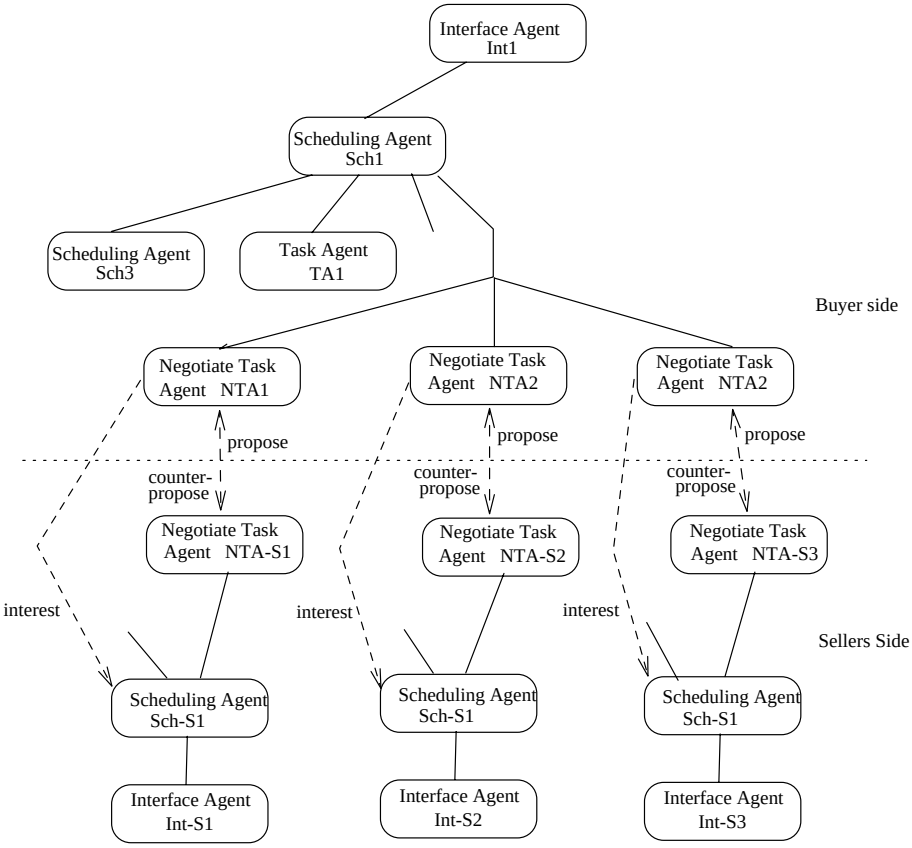


Figure 13. Interactions of Negotiate Task Agents

At the seller side, there is another task, namely 'deliver' to deliver the purchased product to the customer. This task may be a user task, that is, a user task agent may be necessary to execute the task. This requires another turn of auctions. There might be many postmen who can deliver this item. They all have interface agents running on behalf of them. One of the interface agents gets the job as a result of an auction and displays the work to be done (item properties and address of delivery, delivery time, etc.) as a work item of the worklist of that postman.

Example 4.1

The example given in this section describes a scenario for an active buyer. The scenario is as follows: "There is a buyer who wants to buy a computer and a desk for the computer. He also needs three game CDs." Whenever, the customer (buyer) connects to the marketplace (by contacting a URL) an interface agent becomes active to interact with the user. The interface agent obtains the request from the buyer as depicted in Figure 6a. In the window, the details for computer is shown but the details of the desk and CDs are also gathered. Afterwards, the

interface agent converts this request into an initial workflow process definition in XML (see Appendix C) by adapting the workflow template (Figure 7) obtained from the marketplace directory for an active buyer. Graphical representation of the initial workflow process definition that contains only the buyer requirements is shown in Figure 14.

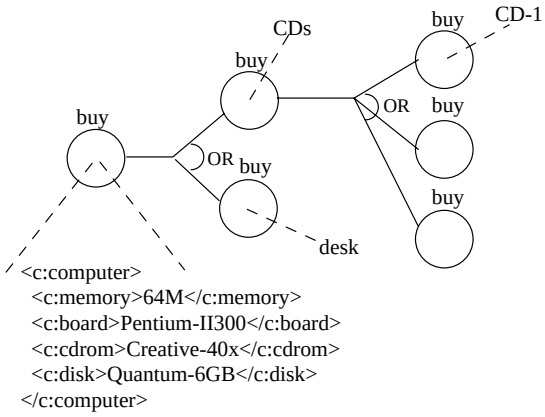


Figure 14. Workflow Process of the buyer Request

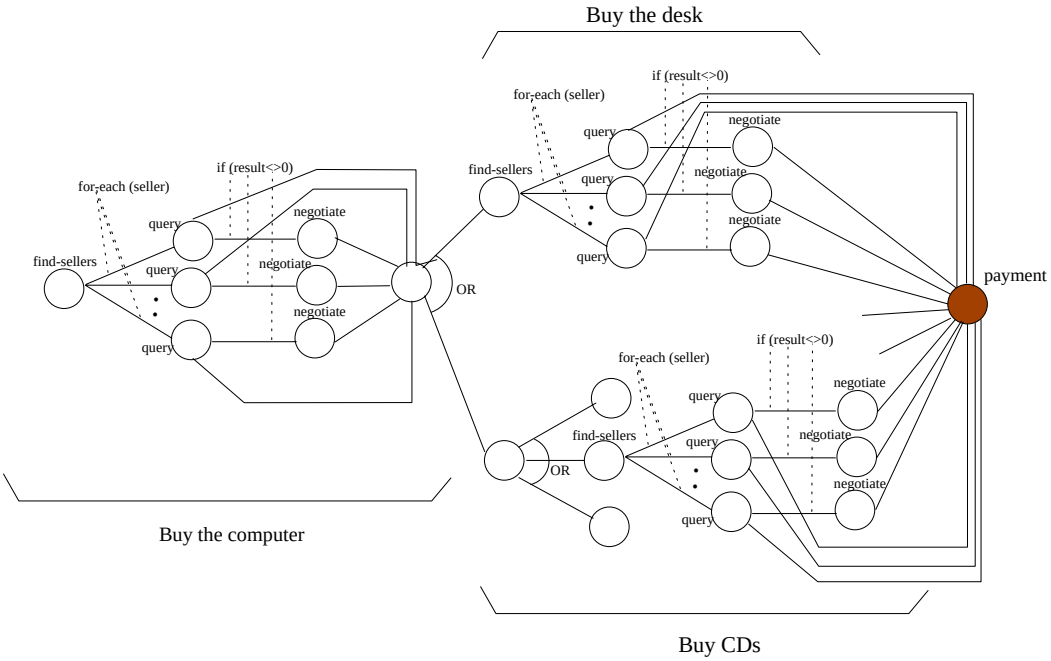


Figure 15. Adapted Workflow Process

Figure 15 illustrates the final workflow definition as adapted from the template using the definition of the buyer request given in Figure 14.

The other steps are executed as described in the scenario.

Related Work

There are several marketplace architectures proposed and some of them are being commercially used.

One of the early marketplaces is Kasbah [A. Chaves and P. Maes, 1996]. Kasbah is a Web-based system where the users create autonomous agents that buy and sell goods or services on their behalf. Selling agents try to sell themselves by going into the marketplace contacting interested buying agents and negotiating with them to find the best deal. The marketplace's job is to facilitate interaction between the agents. In Kasbah, a specific communication language is used since their agents are locally built and can communicate via a predefined set of methods.

In [M. Tsvetovatyy et al., 1997], an architecture (MAGMA) for a marketplace that includes the infrastructure required for conducting commerce on the Internet is proposed. MAGMA supports communication among agents and allows for various forms of automated and human-controlled transactions. The Vickrey mechanism ([W. Vickrey, 1961]) is implemented as the negotiation strategy. There are several trader agents, an Advertising server and a Bank in MAGMA. Trader agents are responsible for buying and selling goods. The Advertisement server provides a classified advertisement that includes search and retrieval of ads by category. Finally, the Bank provides a set of basic banking services such as checking accounts and electronic cash. All agents communicate with each other through socket connections.

OFFER [M. Bichler et al., 1998] is an electronic brokering architecture which uses OMG's CORBA as a distribution infrastructure. There are three main components: suppliers, customers and e-brokers. A customer can search for a service either directly in the e-catalog of the supplier or use the e-broker to search all the e-catalogs of all the suppliers, which are registered with this broker. CORBA is chosen as the communication infrastructure to solve the interoperability problem. As the negotiation mechanism e-brokers employ simple auction implementations.

EMP [S. Boll et al., 1999] is a marketplace that has a DBMS based architecture. Business transactions within the electronic market are realized by a set of modular market services like offering, buying, registration, authentication, etc. Product data are stored in a DBMS and accessed by the market server through JDBC (Java DataBase Connectivity) interface.

In [B. Reich and I. Ben-Shaul, 1998], Global Electronic Market (GEM) system developed using Java is described. It provides a generic market framework and infrastructure along with the specifications of component interfaces that need to be implemented and plugged into the framework in order to obtain an operational market.

In [A. Dogac et al., 1998c], we present some initial ideas on a workflow based electronic marketplace on the Web.

In [K. Decker et al., 1996], the notions of agent matchmaking and brokering behaviors are defined. KQML performatives are used in describing roles and interactions among agents. Some experiments are conducted to evaluate performance tradeoffs of matchmade and brokered systems. The authors conclude that the brokered systems allow efficient load balancing and have lower overhead while matchmade systems are more robust and retain dynamic naming capabilities, which are required in marketplaces.

In [K. Sycara and D. Zeng, 1996], an architecture for coordinating multiple intelligent software agents is presented. The authors classify the agents as interface agents, task agents and information agents. Interface agents are those whose main task is information filtering to alleviate the user's cognitive overload. Task agents help users perform tasks by formulating problem solving plans and carrying out these plans through querying and exchanging information with other agents. Information agents provide access to a possibly heterogeneous collection of resources. The architecture suggested in this paper has influenced the component-based design of MOPPET.

In [Q. Chen et al., 1998], a dynamic-agent infrastructure which supports dynamic behavior modification of agents is proposed. Functions and actions of the dynamic agent are not predefined but can be loaded and modified at run-time. The architecture supports mobility of the agents. The authors also present how service provisioning can be realized through the use of dynamic agents.

There is some previous work on realizing a workflow system with the use of agents. DartFlow workflow management system [T. Cai et al., 1997] uses Web-browser embedded Java applets as its front end and transportable agents as the backbone. A transportable agent is a program that migrates machine to machine in a heterogeneous network. In DartFlow, each business process can be handled by an agent. Agent Tcl system is used to implement transportable agents. Since agents in DartFlow do not use a standard communication language, its usage is limited to those who make use of Agent Tcl system.

[M. N. Huhns and M. P. Singh, 1998] presents the usage of co-operating agents to manage heterogeneous transaction workflows. Agents communicate through a common ontology to realize the parts of a transaction. There are two kinds of computational agents: actors and knowledge-based systems. The actors are used to control interactions among the components of the architecture. The knowledge-based agents are used when reasoning is needed such as deciding what tasks should be performed next.

In [J. Miller et al., 1998], the use of Web technology for workflow is presented with METEOR$_2$ Web-based workflow management system (WebWork). WebWork is said to be web-based rather than web-enabled since both interfaces and communication/distribution infrastructures are built using Web technology. Data flow is realized through exchanging HTML pages and CGI is the main communication mechanism with servers.

[N. R. Jennings et al., 1996] designs and implements the business process management using an agent-based approach. In this work, the business process is viewed as a collection of autonomous problem solving entities that negotiate with one another to coordinate their sub-activities. Agents communicate through a

specific Agent Communication Language (ACL) which is only interpretable by the native agents.

While [N. R. Jennings et al., 1996] emphasizes the negotiation aspects, [T. Tesh and K. Aberer, 1998] focuses on how agreements among agents can be enforced without a central control. The authors present a formal framework for contract management among autonomous agents. The agents tell a contract manager their interests in the deal and the contract manager is responsible for forming a contract acceptable to both parties and to schedule the exchanges in a way that considers the individual interests.

Conclusions

The electronic commerce process models developed thus far like e-shops, e-malls [P. Timmers, 1998] imitate their off-line counterparts. However to further accelerate the diffusion of electronic commerce, the opportunities and technologies offered by the electronic medium should be fully exploited to provide the users with better commerce environments. The complexities of commerce processes must be hidden from the users through appropriate exploitation of advanced technologies. Providing an open and interoperable architecture is also very important.

This paper proposes an architecture that addresses these issues by providing the users with an able and user friendly environment to express their needs and the system handles the underlying complexities through an agent-based workflow architecture. MOPPET architecture is also based on an open and interoperable infrastructure namely XML and CBL.

MOPPET is currently being implemented. Agents are implemented to use KQML parser of JATLite from Stanford University which follows the full KQML grammar both for standard and extended performatives [JatLite, 1999]. JATLite also provides mechanisms for communication among agents. For parsing XML documents (contents of KQML messages), agents make use of DataChannel's

XJParser which is a validating (using both DTDs and XML data) XML parser together with a DOM implementation [XJParser, 1999].

References

A. Chaves and P. Maes (1996). Kasbah: An agent marketplace for buying and selling goods. In Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK.

A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu (1998). XML-QL: A query language for XML. W3C Document, http://www.w3.org/TR/NOTE-xml-ql.

A. Dogac (1998). Guest Editor. ACM Sigmod Record Special Section on Electronic Commerce. 27(4), December.

A. Dogac (1999). Guest Editor. Distributed and Parallel Databases, Special Issue on Electronic Commerce. Kluwer. to appear.

A. Dogac, E. Gokkoca, S. Arpinar, P. Koksal, I. Cingil, B. Arpinar, N. Tatbul, P. Karagoz, U. Halici, and M. Altinel (1998a). Design and implementation of a distributed workflow management system: METUFlow. In A. Dogac, L. Kalinichenko, T. Ozsu, and A. Sheth, editors, *Workflow Management Systems and Interoperability*. Springer-Verlag. NATO ASI Series.

A. Dogac, I. Durusoy, S. Arpinar, E. Gokkoca, N. Tatbul, and P. Koksal (1998b). METU-EMar: An agent-based electronic marketplace on the web. In [C. Nicolaou and C. Stephanidis, 1998].

A. Dogac, I. Durusoy, S. Arpinar, N. Tatbul, P. Koksal, I. Cingil, and N. Dimililer (1998c). A workflow-based electronic marketplace on the web. In [A. Dogac, 1998].

B. Arpinar (1998). *Formalization of Workflows and Correctness Issues in the Presence of Concurrency*. Ph.D. thesis, Middle East Technical University, Dept. of Computer Engineering, November.

B. Arpinar, U. Halici, S. Arpinar, and A. Dogac (1999). Formalization of Workflows and Correctness Issues in the Presence of Concurrency. *Distributed and Parallel Databases*, to appear.

B. Meltzer and R. Glushko (1998). XML and electronic commerce: Enabling the network economy. In [A. Dogac, 1998].

B. Reich and I. Ben-Shaul (1998). A componentized architecture for dynamic electronic markets. In [A. Dogac, 1998].

C. Nicolaou and C. Stephanidis (1998). Editors. Research and Advanced Technology for Digital Libraries, Lecture Notes in Computer Science, Springer.

D. Suciu (1998). Semistructured data and XML. In *Proceedings of International Conference on Foundations of Data Organization*.

DOM (1998). *Document Object Model Level 1 Specification*. W3C Recommendation. http://www.w3.org/TR/REC-DOM-Level-1.

H. Raiffa (1982). The Art and Science of Negotiation. Harward University Press.

JATLite (1999). http://java.stanford.edu/JATLite-index.html

J. Miller, D. Palaniswami, A. Sheth, K. Kochut, and H. Singh (1998). WebWork: METEOR$_2$'s web-based workflow management system. *Journal of Intelligent Information Systems*, 10(2):1--30.

K. Bohm, K. Aberer, E. Neuhold, and X. Yang (1997). Structured document storage and refined declarative and navigational access mechanisms in HyperStorM. *The VLDB Journal*, 6(4):296--311.

K. Decker, M. Williamson, and K. Sycara (1996). Matchmaking and brokering. In *Proc. of the Second International Conference on Multi-Agent Systems (ICMAS-96)*.

K. Sycara and D. Zeng (1996). Coordination of multiple intelligent software agents. *Intl. Journal of Cooperative Information Systems*, 5 (2&3).

M. Bichler, C. Beam, and A. Segev (1998). OFFER: A broker-centered object framework for electronic requisitioning. In *IFIP Conference "Trends in Electronic Commerce '98*.

M. N. Huhns and M. P. Singh (1998). Managing heterogeneous transaction workflows with co-operating agents. In N.R. Jennings and M. J. Wooldridge, editors, *Agent Technology: Foundations, Applications, and Markets*. Springer-Verlag.

M. Tsvetovatyy, M. Gini, B. Mobasher, and Z. Wieckowski (1997). MAGMA: An agent-based virtual market for electronic commerce. *Applied Artificial Intelligence*, 11(6):501--524.

M. Yilmaz (1999), *Design and Implementation of an Agent Architecture for an Electronic Marketplace*, MSc Thesis, Dept. of Computer Eng., Middle East Technical University, March.

N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien, and M. E. Wiegand (1996). Agent-based business process management. *International Journal of Cooperative Information Systems*, 5(2\&3):105--130.

P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu (1996). A query language and optimization techniques for unstructured data. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, Montreal, Canada.

P. Faratin, C. Sierra, and N. R. Jennings (1998). Negotiation decision functions for autonomous agents. *Int. Journal of Robotics and Autonomous Systems*.24 (3-4):159-182.

P. Timmers (1998). Business models for electronic markets. In: Y. Gradient, B. F. Schmid, D. Selz: EM-Electronic Commerce in Europe. EM-Electronic Markets, 8 (2), July http://www.ispo.cec.be/ecommerce/busimod.htm.

Q. Chen, P. Chundi, U. Dayal, and M. Hsu (1998). Dynamic-agents for dynamic service provisioning. In *Proc. of 3rd Intl. Conf. on Cooperative Information Systems*, NewYork.

RDF (1998). *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Working Draft. http://www.w3.org/TR/WD-rdf-syntax.

RDFSchema (1998). *Resource Description Framework (RDF) Schema Specification*. W3C Working Draft. http://www.w3.org/TR/WD-rdf-schema.

RosettaNet (1998). http://www.rosettanet.org/general/finished-project/laptop.html.

S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener (1997). The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1).

S. Boll, A. Gruner, A. Haaf, and W. Klas (1999). EMP-a database driven electronic marketplace for business-to-business commerce on the internet. In [A. Dogac, 1999]. to appear.

T. Cai, P. A. Gloor, and S. Nog (1997). DartFlow: A workflow management system on the web using transportable agents. Technical report, Dartmouth College.

T. Tesh and K. Aberer (1998). Scheduling non-enforceable contracts among autonomous agents. In *3rd Intl. Conf. on Cooperative Information Systems (COOPIS'98)*}, NewYork.

TOS (1997). *Trading Object Service*. OMG Document.

VEO Systems Inc. (1998). http://www.veosystems.com.

W. Vickrey (1961). Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(8):8--37.

XJParser (1999). http://xdev.datachannel.com/downloads/xjparser.

XML (1998). *Extensible Markup Language (XML) 1.0*. W3C Recommendation. http://www.w3.org/TR/REC-xml-19980210.

Y. Bakos (1998). The Emerging Role of Electronic Marketplaces on the Internet, *Communications of the ACM*, 41 (8): 35-42, August.

Y. Labrou and T. Finin (1994). A semantics approach for KQML - a general purpose communication language for software agents. In *Third Intl. Conf. on Information and Knowledge Management (CIKM'94)*.

Y. Labrou and T. Finin (1997). A proposal for a new KQML specification. Technical Report TR-CS-97-03, University of Maryland, Baltimore County.

Appendix A

DTD for a Workflow Process Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE simple [
<!ENTITY % activity '(task,retry?,undo?,compensation?)|assignment|block|
iterative-block|conditional-block|for-each-block'>

<!-- Top level Container: workflows -->
<!ELEMENT workflows (#PCDATA,process+)>
<!ELEMENT process (variables?,parameter*,%activity;*)>
<!ATTLIST process
    name       CDATA  #REQUIRED
    duration   CDATA  #IMPLIED >
<!ELEMENT variables (var*)>
<!ELEMENT var (#PCDATA | list)>
<!ATTLIST var
    mode (INTERNAL|EXTERNAL) "INTERNAL"
    type       (INT|STRING|XML|XML-QL|LIST)>
<!ELEMENT parameter (#PCDATA,content?)>
<!ATTLIST parameter
    mode       (IN|OUT)>
<!ELEMENT task (parameter*)>
<!ATTLIST task
    name       CDATA  #REQUIRED
    type       (TASK|SUBPROCESS) "TASK"
    role       CDATA  #IMPLIED
    user       CDATA  #IMPLIED
    description CDATA #IMPLIED>
<!ELEMENT retry (condition?)>
<!ATTLIST retry
    number     CDATA #REQUIRED>
<!ELEMENT undo (condition?,%activity;*)>
<!ELEMENT compensation (%activity;*)>
<!ELEMENT assigment ((lhs-expr|list-element),rhs-expr)>
<!ELEMENT block (%activity;*, compensation?)>
<!ATTLIST block
    type       (SERIAL|AND-PARALLEL|OR-PARALLEL|XOR-PARALLEL|CONTINGENCY)
"SERIAL"
    name       CDATA #IMPLIED >
```

```
<!ELEMENT iterative-block (condition,%activity;*,compensation?)>
<!ELEMENT conditional-block (condition,%activity;*,compensation?,else?)>
<!ELEMENT else (%activity;,compensation?)>
<!ELEMENT for-each-block (list,%activity;*,compensation?)>
<!ATTLIST for-each-block
    type      (AND-PARALLEL|OR-PARALLEL|XOR-PARALLEL) "AND-PARALLEL">
<!ELEMENT condition(lhs-expr,comparison-operator,rhs-expr)>
<!ELEMENT lhs-expr (#PCDATA)>
<!ELEMENT comparison-operator EMPTY>
<!ATTLIST comparison-operator
    type      (EQUALITY|GREATER|LESS|GEQ|LEQ|NOTEQUAL)>
<!ELEMENT rhs-expr ((int|list|string),(operator,(int|list|string))*>
<!ELEMENT operator ("+"|"-"|"*"|"/"|"%")>
<!ELEMENT int (#PCDATA)>
<!ELEMENT string (#PCDATA)>
<!ELEMENT list (#PCDATA)>
<!ATTLIST list
    type      (INT|STRING|XML|XML-QL) "STRING" >
<!ELEMENT list-element (#PCDATA,index,content)>
<!ELEMENT index (int) >
<!ELEMENT content ANY>

]>
```

Appendix B

Workflow Templates

Active Buyer Template

```
<workflows>wf-temp1
 <process name='active-buyer'}
  <parameter mode=IN>items</parameter>
  <variables>
    <var type=XML-QL>result</var>
    <var type=LIST><list type=XML-QL>items</list></var>
    <var type=LIST><list type=STRING>sellers</list></var>
  </variables>
  <block>
   <for-each-block type=AND-PARALLEL>
    <list>items</list>
    <task name='find-sellers' description='http://www.srdc.metu.edu.tr/mpd/find-sellers.RDF'>
      <parameter mode=IN><list-element>items<index><int>i</int></index>
                  </list-element></parameter>
      <parameter mode=OUT>sellers</parameter>
    </task>
    <for-each-block type=AND-PARALLEL>
      <list>sellers</list>
      <task name='query' description='http://www.srdc.metu.edu.tr/mpd/query.RDF'>
        <parameter mode=IN><list-element>items<index><int>i</></index>
                  </list-element></parameter>
        <parameter mode=IN><list-element>sellers<index><int>j</int></index>
                    </list-element></parameter>
        <parameter mode=OUT>result</parameter>
      </task>
      <conditional-block> <condition>... </condition>
        <task name='negotiate' ...> ... </task>
      </conditional-block>
    </for-each-block>
   </for-each-block>
   <task name='payment' type=SUBPROCESS ...> ... </task>
  </block>
 </process>
 ...  (definitions of other processes eg. payment)
</workflows>
```

Passive Buyer Template

```
<workflows>wf-temp2
 <process name='passive-buyer'>
  <variables>
   <var mode=EXTERNAL type=INT>request</var>
   <var mode=EXTERNAL type=XML>product-info</var>
  </variables>
  <conditonal-block>
   <condition>
    <lhs-expr>request</lhs-expr><comparison-operator type=EQUALITY/>
    <rhs-expr><int>1</int></rhs-expr>
   </condition>
   <task name='user-confirmation' description='http://www.srdc.metu.edu.tr/mpd/user-confirmation.RDF'>
    <parameter mode=IN>product-info</parameter>
    <parameter mode=OUT>reply</parameter>
   </task>
   <conditional-block>          (if reply is OK (1))
    <condition> ... </condition>
    <task name='negotiate' ...> ... </task>
   </conditional-block>
   <task name='payment' type=SUBPROCESS ...> ... </task>
 </process>
 ...
</workflows>
```

Active Seller Template

```
<workflows>wf-temp3
 <process name='active-seller'>
  <variables>
   <var type=XML>product-info</var>
   <var type=LIST><list type=STRING>buyers</list></var>
  </variables>
  <block>
   <task name='get-product-info' description='http://www.srdc.metu.edu.tr/mpd/get-product-info.RDF'>
    <parameter mode=OUT>product-info</parameter> </task>
   <task name='find-buyers' ...> <parameter mode=OUT>buyers</parameter> </task>
   <for-each-block type=AND-PARALLEL>
    <list>buyers</list>
```

```
      <task name='send-request' description='http://www.srdc.metu.edu.tr/mpd/send-request.RDF'>
        <parameter mode=IN>product-info</parameter>
        <parameter mode=OUT>reply</parameter>
      </task>
      <conditional-block>          (if reply is positive)
        <condition> ... <condition>
        <task name='negotiate' ...> ... </task>
      </conditional-block>
    </for-each-block>
    <task name='payment' type=SUBPROCESS ...> ... </task>
    <task name='deliver' role='postman' ...> ... </task>
  </block>
 </process>
 ...
</workflows>
```

Passive Seller Template

```
<workflows>wf-temp4
 <process name='passive-seller'>
  <variables>
   <var mode=EXTERNAL type=INT>request</var>
   <var mode=EXTERNAL type=XML-QL>query</var>
   <var mode=EXTERNAL type=INT>interest</var>
   <var type=XML-QL>result</var>
  </variables>
  <conditonal-block>
   <condition>              (if request is 1)
    <lhs-expr>request</lhs-expr><comparison-operator type=EQUALITY/>
    <rhs-expr><int>1</int></rhs-expr>
   </condition>
   <task name='query-catalog' description='http://www.srdc.metu.edu.tr/mpd/query-catalog.RDF'>
    <parameter mode=IN>query</parameter>
    <parameter mode=OUT>result</parameter>
   </task>
   <conditional-block>              (if interest is 1)
    <condition> ... </condition>
    <task name='negotiate' ...> ... </task>
   </conditional-block>
   <task name='payment' type=SUBPROCESS ...> ... </task>
```

```
  </process>
  ...
</workflows>
```

Appendix C

Adapted Workflow Template

```
<workflows>WF1
 <process>
  <variables>
   <var type=XML-QL>result</var>
   <var type=LIST><list type=STRING>sellers</list></var>
   <var type=XML>computer-info</var>
   <var type=XML>desk-info</var>
   <var type=XML>CD-info1</var>
   ...
  </variables>
  <block>
   <task name='find-sellers' description='http://www.srdc.metu.edu.tr/mpd/find-sellers.RDF'>
    <parameter mode=IN>computer-info
          <content>
            <?xml namespace name='http://www.srdc.metu.edu.tr/mpd/computer.dtd' as "c"?>
                <c:item><c:computer>
                <c:memory>64M</c:memory>
                <c:board>Pentium-II300</c:board>
                <c:cdrom>Creative-40x</c:cdrom>
                <c:disk>Quantum6GB</c:disk>
                </c:computer></c:item>
          </content> </parameter>
    <parameter mode=OUT>sellers</parameter>
   </task>
   <for-each-block type=AND-PARALLEL>
    ...
    <task name='query' description='http://www.srdc.metu.edu.tr/mpd/query.RDF'>
     <parameter mode=IN>computer-info<content> ... </content>
     </parameter>
     ...
    </task>
    ...
   </for-each-block>
   <block type=OR-PARALLEL>
    <task name='find-sellers' ...>
     <parameter mode=IN>desk-info .... </parameter>
    </task>
```

```
        ...
        <block type=OR-PARALLEL>
          <task name='find-sellers' ...> <parameter mode=IN>CD-info1 ... </parameter> </task>
          ...
        </block>
      </block>
      <task name='payment'....> ... </task>
    </block>
  </process>
</workflows>
```