Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation

Qiming Chen, Meichun Hsu, Umeshwar Dayal, Martin Griss

HP Labs
1501 Page Mill Road, MS 1U4
Palo Alto, California, CA 94303, USA
+1-650-857-3060

{qchen,mhsu,dayal,griss}@hpl.hp.com

ABSTRACT

E-Commerce is a distributed computing environment with dynamic relationships among a large number of autonomous service requesters, brokers and providers. Agents with predefined functions but without the ability to modify behavior dynamically may be too limited for mediating E-Commerce applications properly, since they cannot switch roles or adjust their behavior to participate in dynamically formed partnerships.

We have developed a Java based *dynamic agent* infrastructure for E-Commerce automation, which supports *dynamic behavior modification* of agents, a significant difference from other agent platforms. Supported by dynamic agents, mechanisms have been developed for plugging in workflow and multi-agent cooperation, and for supporting dynamic workflow service provisioning that allows workflow services to be constructed on the fly.

XML is chosen as our agent communication message format. Since different problem domains have different ontology, we allow agents to communicate with domain specific performatives and act using corresponding interpreters. Dynamic agents can carry, switch and exchange interpreters. Our approach enables document-driven agent cooperation and DTD based program generation, and further, allows agents to exchange and share ontology for multiple or even dynamic domains. In this way, the cooperation of dynamic agents supports *plug-and-play commerce*, mediating businesses that are built on one another's services. A prototype has been developed at HP Labs.

Keywords

Dynamic agents, workflow, XML

1. INTRODUCTION

This work focuses on providing a multi-agent cooperation infrastructure to support E-Commerce automation. Before discussing our solutions, we first present a typical E-Commerce scenario and the requirements for E-Commerce automation.

1.1 E-Commerce Automation

E-Commerce applications operate in a **dynamic and distributed environment**, dealing with a large number of heterogeneous information sources with *evolving contents* and *dynamic availability* [24]. They typically rely on distributed and autonomous tasks for information search, fusion, extraction and processing, without centralized control.

An E-Commerce scenario (Figure 1) typically involves the following activities: identifying requirements; brokering products; brokering vendors; negotiating deals; or making purchase and payment transactions. Today, these activities are initiated and executed by humans. In the future, we see them being conducted by software agents.

Software agents are personalized, continuously running and semi-autonomous, driven by a set of beliefs, desires and intentions (BDI). They can be used to *mediate* users and servers to automate a number of the most time-consuming tasks in E-Commerce, with enhanced parallelism [3,16,18,19,20,22]. Agents can also be used for *business intelligence*, such as discovering patterns (e.g. shopping behavior patterns or service providing patterns) and react to pattern changes. For example, suppose the sales of VCRs had been strongly associated with the sales of TVs, but this association has recently weakened as TV buyers turn to buying DVDs instead of VCRs. Such a change in the association helps to explain or predict the slow down of VCR sales. Moreover, agents can selectively preserve

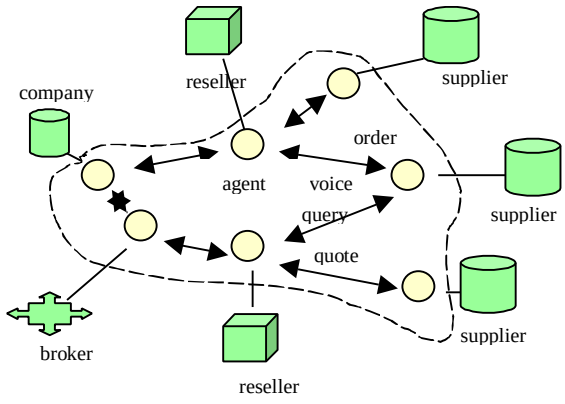

Figure 1: E-C Automation through Multi-agent Cooperation

data and themselves become *dynamic information sources*.

E-Commerce is also a **plug and play environment**. Business processes and agent cooperation are embedded in each other. Services need to be created dynamically on demand. Business

partnerships (e.g. between suppliers, resellers, brokers, and customers) need to be created dynamically and maintained only for the required duration such as a single transaction. Agents need to flexibly switch roles and adjust their capabilities to participate in dynamic business partnerships. Furthermore, agents may cooperate in different application domains.

The dynamic nature of E-Commerce requires multi-agent cooperation to be based on **dynamic ontology**. By dynamic ontology we mean that the concepts, rules and facts underlying agent interaction, are different from domain to domain, and vary from time to time. In order to automate agent cooperation, it is necessary to provide a standard format for encoding messages with meaningful structure and semantics, as well as domain ontology that agents can readily exchange and interpret. This format should be common for agent communication as well as for E-Commerce data exchange in general. The extensible markup language, XML [2], is becoming the standard for data interchange on the Web. We use XML for the above purpose.

Business processes, or workflows [7,9], may be considered as a kind of multi-agent cooperation, in the sense that software agents may be used to perform tasks of business processes, and workflow may be used to orchestrate or control the interactions between agents[15,17,19]. We envisage the need for dynamically plugging them into each other.

At HP labs, we have developed a Java based **dynamic agent** infrastructure for E-Commerce which supports **dynamic behavior modification** of agents, a significant difference from other agent platforms [4,5]. A dynamic agent does not have a fixed set of predefined functions, but instead, it **carries application-specific actions**, which can be loaded and modified on the fly. This allows a dynamic agent to adjust its capabilities and play different roles to accommodate changes in the environment and requirements. Through messaging, dynamic agents can expose their knowledge, abilities and intentions, present requests and exchange objects. They can move to the appropriate location for high-bandwidth conversation. They can also manage their own resources across actions. Such an infrastructure supports dynamic service construction, modification and movement, and allows a dynamic agent to participate in multiple applications and dynamically formed partnerships. With these features, dynamic agents fit well into the dynamic E-Commerce environment.

A multi-agent cooperation infrastructure is developed for E-Commerce automation, where dynamic agents perform various market activities, cooperating through exchanging data as well as programs, switching roles and forming ***dynamic partnership*** that exists only when needed. For example, the agents reselling products, the agents supplying products and the agents providing brokering services for negotiating service terms, etc, may form dynamic partnership for a specific business application. In this way, dynamic agents cooperatively support ***plug-and-play commerce***, allowing businesses to be built on one another's services.

Dynamic agents communicate using XML, and can **dynamically load and exchange different ontologies and XML interpreters** for tasks in different domains.

Finally, the mechanisms for **plugging workflow in agent cooperation,** and **plugging agent cooperation in tasks** of business processes, are introduced. In particular, **dynamic workflow service provisioning** is supported, allowing workflow servers to be built on the fly.

These approaches allow us to provide *a* unified application carrier architecture, a unified agent communication mechanism, a unified way of data flow, control flow and even program flow, but flexible application switching capability, for supporting E-Commerce.

1.2 Related Work

E-Commerce applications operate in a distributed computing environment, where the use of interface-based infrastructures such as CORBA [8] is rather popular. However, interface-based distributed computing is static, in the sense that a service is configured at a "well-known" but fixed location; has pre-defined function (as an implementation of its interface definition such as IDL specification); and such an implementation may not necessarily be portable or movable. The remote function invocation mechanisms are generally based on data-flow, namely, sending requests to and getting results from servers at fixed locations. The flow of programming objects is not supported.

Many existing agent platforms such as Odyssey, Voyager, etc [1,23,25], also lack dynamic-modifiability of behavior, in the sense that an agent must be statically coded and launched for doing only a fixed set of things. For example, an Odyssey agent or a Voyager agent, such as "buyer" or "seller", is pre-coded for that functionality and cannot be changed after launching. Although it can move, meet, or receive messages, it cannot dynamically load new functions to alter its pre-defined behavior. Alternative functionality can only be introduced by launching additional agents. Since it lacks data, knowledge and program management facilities, it lacks the potential to be used across applications.

Some multi-agent systems such as described in [24], use agents with predefined functions, such as query agents, resource agents, etc. Although these systems provide several promising features, they have limited flexibility in highly dynamic E-Commerce oriented cooperative problem solving. This is because they do not provide self-installable and self-configurable system components to act at the appropriate time and location, to adjust their behaviors on the fly for accommodating environment changes, and to exchange program modules for cooperation.

The relationships between multi-agent cooperation and workflow have been studied by us and others [15,17,19]. However, our approach to provide workflow services on the fly during multi-agent cooperation, is unique.

We share the same view as [14] on the importance of XML for E-Commerce. In fact, several agent communication languages such as FIPA[13], KQML[12,21], etc, have been converted to simple XML form. However, ontology varies from domain to domain, and if a problem domain is dynamically formed based on a specific application, its ontology is also dynamic. In KQML and FIPA, domain specific ontology is explicitly supported by using ":ontology" parameter, but no interpreter

switching mechanism provided. Thus we emphasize the need for exchanging domain specific ontology, particularly XML interpreters to allow dynamic agents to participate in multiple applications, to switch domains and to form dynamic partnerships.

The rest of this paper is organized as follows. Section 2 outlines the dynamic agent infrastructure. Section 3 describes the use of XML messaging with dynamic ontology to support multi-agent cooperation. Section 4 shows the way to plug-in workflow to multi-agent cooperation, and vice versa. Finally in section 5, some concluding remarks are given.

2. DYNAMIC AGENTS

To approach E-Commerce automation, agents need to have dynamic behavior while maintaining its identity and consistent communication channels, as well as retaining data, knowledge and other system resources to be used across applications.

It is neither sufficient for an agent to be equipped with a fixed set of application-specific functions, nor practical for the above capabilities to be developed from scratch for each agent. This has motivated us to develop a *dynamic-agent* infrastructure [4,5]. The infrastructure is Java-coded, platform-neutral, light-weight, and extensible. Its unique feature is the support of *dynamic behavior modification* of agents; and this capability differentiates it from other agent platforms and client/server-based infrastructures.

A dynamic-agent has a *fixed part* and a *changeable part*.(Figure 2) As its fixed part, a dynamic-agent is provided with light-weight, built-in management facilities for distributed communication, object storage and resource management. A dynamic agent is capable of carrying data, knowledge and ***programs*** as objects, and executing the programs. The data and programs carried by a dynamic agent form its changeable part. All newly created agents are the same; their application-specific behaviors are gained and modified by dynamically loading Java classes representing data, knowledge and application programs. Thus dynamic-agents are general-purpose **carriers** of programs, rather than individual and application-specific programs.

The architecture of dynamic-agent can be explained in more detail by the following.

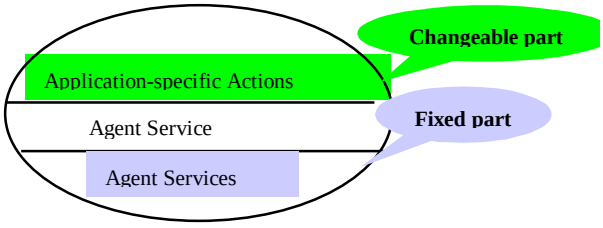

Figure 2: Dynamic Agent

Built-in Facilities. Each dynamic-agent is provided with several light-weight built-in facilities for managing messaging, data and program object storage, action activation, GUI, etc. A carried application, when started, is offered a reference to the underlying built-in management facilities, and can use this reference to access the APIs of the facilities.

A *message-handler* is used to handle message queues, sending, receiving and interpreting inter-agent messages. The interaction styles include one-way, request/reply, and publish/subscribe(selective broadcast). Message forwarding is also supported. An *action-handler* is used to handle the message-enabled instantiation and execution of application programs (Java classes). One dynamic-agent can carry multiple action programs. An *open-server-handler* is used to provide a variety of continuous services, which can be started and stopped flexibly at dynamic-agent run-time. A *resource-handler* is used to maintain an object-store for the dynamic-agent, that contains application-specific data, Java classes and instances including language interpreters, addresses and any-objects (namely, instances of any class).

Applications executed within a dynamic-agent use the built-in dynamic-agent management facilities to access and update application-specific data in the object-store, and to perform inter-agent communication through messaging.

Dynamic Behavior. Enabled by corresponding messages, a dynamic-agent can load and store programs as Java classes or object instances, and can instantiate and execute the carried programs. Within these programs, built-in functions can be invoked to access the dynamic-agent's resources, activate and communicate with other actions run on the same dynamic-agent, as well as communicate with other dynamic-agents or even stand-alone programs.

Mobility. Two levels of mobility are supported. A dynamic-agent may be moved to or cloned at a remote site. Programs carried by one dynamic-agent may be sent to another, to be run at the receiving site.

Coordination. Every dynamic-agent is uniquely identified by its symbolic name. Similar to FIPA [13], a coordinator agent is used to provide naming service, mapping the name of each agent to its current socket address. The coordinator is a dynamic-agent with the added distinction that it maintains the agent name registry and, optionally, resource lists. When a dynamic-agent, say ***A***, is created, it will first attempt to register its symbolic name and address with the coordinator by sending a message to the coordinator. Thereafter, ***A*** can communicate with other dynamic-agents by name. When ***A*** sends a message to another whose address is unknown to ***A***, it consults the coordinator to obtain the address. If ***A*** is instructed to load a program but the address is not given, it consults the coordinator or the request sender to obtain the address. Each dynamic-agent also keeps an address-book, recording the addresses of those dynamic agents that have become known to it, and are known to be alive.

In a multi-agent system, besides naming service, other coordination may be required, and provided either by the coordinator or by other designated dynamic-agents that provide brokering services. A ***resource-broker*** maintains a hierarchically structured agent capability registry. The leaf-level nodes referring to the dynamic agents carry corresponding programming objects (action modules). Agents contact the resource-broker when acquiring or dropping new programming objects, and when they need a program such as a domain specific XML interpreter. A ***request-broker*** is used to isolate

the service requesters from the service providers (i.e. dynamic-agents that carry the services) allowing an application to transparently make requests for a service. An ***event-broker*** delivers events, treated as asynchronous agent messages, to event subscribers from event generators. Event notification may be point-to-point, where the event subscribers know the event generators and make the subscriptions accordingly; or multicast, where one or more event-brokers are used to handle events generated and subscribed anywhere in the given application domain. These event distribution mechanisms allow agents to subscribe to events without prior knowledge of their generators.

Dynamic-agents may form hierarchical groups, with a coordinator for each group. Based on the group hierarchy a multilevel name service can be built.

Dynamic Role Assignment and Switching. A role is the abstraction of one or more agents, providing a dynamic interface between agents and cooperative processes. Agents playing a specific role follow the normative rules given by the cooperative process. The action carrying capability, together with other capabilities of dynamic agents, make the mapping from a role to a specific agent extremely simple, and allows one agent to play different roles, even simultaneously. In fact, a dynamic agent obtains the capability for playing a role simply by downloading the corresponding programs, passes a role to another agent by uploading the programs, or switches roles by executing different programs it carries.

In order for an agent to switch its roles back and forth, the agent must be equipped with memory capability across multiple activities, and possess a consistent communication channel. For example, an agent responsible for ordering a product may well be the one responsible for order cancellation. Such history sensitivity and identity consistency are exactly the strength of our dynamic agents.

For example, adjusting load balance by task reallocation is a kind of agent cooperation. Reallocation is beneficial when tasks are not initially allocated to the agents that handle them least expensively. Each agent can contract out tasks it had previously contracted in. give out if the task is more suitable for another agent to perform, or accept a task if the task is more suitable to be done by itself than by another. The action carrying and exchanging capability of dynamic agents naturally support task reallocation.

Dynamic agents form a dynamic distributed computing environment for E-Commerce. In the following sections we shall show our solutions to E-Commerce automation that take advantage of this.

3. MULTI-AGENT COOPERATION WITH XML MESSAGING

Autonomous agents cooperate by sending messages and using concepts from a domain ontology. A standard message format with meaningful structure and semantics, and a mechanism for agents to exchange ontologies and message interpreters, have become key issues. Furthermore, the message format should be accepted not only by the agent research community, but also by all information providers.

3.1 Document-driven Agent Cooperation

The Internet and Web represent an increasingly important channel for B2B (business to business), B2C (business to consumer) and C2C (consumer to consumer) E-Commerce. As the extensive markup language, XML, is fast becoming the standard for data interchange on the Web [13,14], we chose XML as the primary message format for dynamic agent communication (Figure 3).

Dynamic agents send and receive information through XML encoded messages. We use a KQML/FIPA ACL-like format, encoded in XML, e.g.

```
<MESSAGE type="REQUEST" from="XYZ" to="HPSV"
 interpreter="xml_shopping">
   <CONTENT>
      <ORDER> a xml document </ORDER>
   </CONTENT>
</MESSAGE>
```

XML tags markup the information and break up the data into parts, In XML based messages, agents encode information with meaningful structure and commonly agreed semantics. On the receiving side, different parts of the information can be identified and used in different applications.

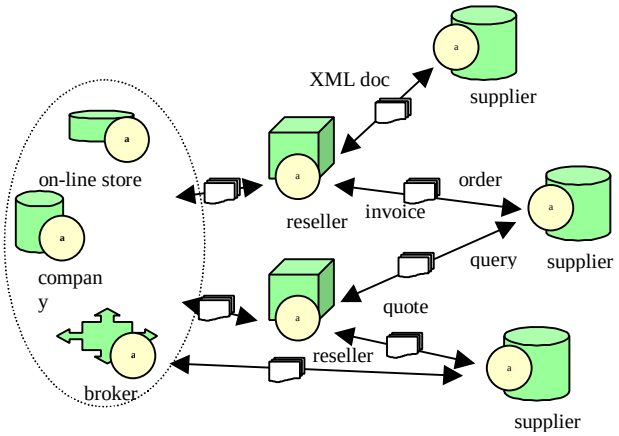

Figure 3: Multi-Agent Cooperation with XML Messaging

In fact, an XML document is an **information container** for reusable and customizable components, which can be used by any receiving agent This is the foundation for document-driven agent cooperation. By making Web accessible to agents with XML, the need for customer interfaces for each consumer and supplier will be eliminated. Agents may use XML format to explain their BDI, explaining new performatives by existing, mutually understood ones. Based on the commonly agreed tags, agents may use different style DTDs to fit the taste of the business units they mediate. Further, a dynamic agent can carry an XML front-end to a database for data exchange, where both queries and answers are XML encoded.

The power of XML, the role of XML in E-Commerce, and even the use of XML for agent communication, have been recognized. Although XML is well structured for encoding semantically meaningful information, it must be based on an ontology. As ontology varies from domain to domain, and

dynamic for dynamically formed domains, The more significant issue is to exchange the semantics of domain models, and interpret messages differently in different problem domains.

Generally speaking, domain ontology provides a set of concepts, or meta-data, that can be queried, advertised and used to control the behavior of agent cooperation. These concepts can be marked using XML tags, and then a set of commonly agreed tags, underlie message interpretation. The structures and the semantics of the documents used in a particular problem domain are represented by the corresponding DTDs and interpreters.

We use different languages, all in XML format, for different problem domains, such as product ordering, market analysis, etc. Accordingly, we use an individual interpreter for each language. Dynamic agents can exchange those DTDs together with documents, and exchange those interpreters as programming objects, in order to understand each other in communication.

3.2 DTD based Program Generation

Since information sources are evolving, it is unlikely that we can use fixed programs for information accessing and processing. Our solution is to let a dynamic agent carry program tools that generate XML oriented data access and processing programs based on DTDs. A DTD (like a schema) provides a grammar that tells which data structures can occur, and in what sequences. Such schematic information is used to automatically generate programs for basic data access and processing, i.e. creating classes that recognize and process different data elements according to the specification of those elements. For example, from an XML document including tag UNIT_PRICE, it is easy to generate a Java method "getUnitPrice", provided that the meanings of tags are understood, and a XML parser is appended to the JDK classpath. The XML parser we use is the one developed by Sun Microsystems that supports SAX (Simple API for XML) and conforms to W3C DOM (Document Object Model [11]).

The advantage of automatic program generation from DTDs, is allowing tasks to be created on the fly, in order to handle the possible change of document structures. Thus for example, when a vendor publishes a new DTD for its product data sheet, based on that DTD, an agent can generate the appropriate programs for handling the corresponding XML documents. Agents use different programs to handle data provided by different vendors.

3.3 Ontology Model Switching

Different application domains have different ontology models, with different agent communication languages and language interpreters, although they are in XML format. In a particular application domain, agents communicate using domain specific XML language constructs and interpreters.

In our implementation, a dynamic agent can participate in multiple applications. It communicates with other agents for the business of one domain, say D_a, using D_a's language and language interpreter; for the business of another domain, say D_b, using D_b's language and language interpreter. A dynamic agent can carry multiple interpreters. It switches application domains and ontologies by switching the DTD's and interpreters it uses.

We load an interpreter, say, *xml_shopping*, to an agent by sending it a message :

```
<MESSAGE type="LOAD" from="A" to="B"
 interpreter="xml.default">
   <CONTENT>
     <LOAD_INTERPRETER
       class="da.XMLshoppingInterpreter"
       url="file:host.hp.com/Dmclasses">
           xml_shopping
     </LOAD_INTERPRETER>
   </CONTENT>
</MESSAGE>
```

Let us assume that agent ***A*** receives from agent ***S*** a message, say ***e***, that indicates the interpreter as a message attribute. If ***A*** does not have that interpreter, ***A*** will reply by a query message for that interpreter. If the sender has the interpreter, it will then send it to ***A***, in order for ***A*** to understand message ***e***; otherwise the sender will contact the coordinator (or resource manager), for ***A*** to have the interpreter loaded.

This process can be shown by the messages shown in Figure 4.

Fr	To	Interpreter	Message Content
S	A	Xml.shopping	<ORDER> an xml document </ORDER>
A	S	Xml.default	<QUERY type="interpreter"> xml_shopping </QUERY>
S	A	Xml.default	<LOAD type="interpreter"> xml_shopping </LOAD>

Figure 4: Messages Sent to Load an Interpreter

We provide two kinds of basic support for dynamic partnership.

- Agents can form groups dynamically. This is very different from statically formed distributed computing domains such as DCE domains. Since agents in different groups can get each other's address through hierarchical naming service, they can reach each other even if in different "physical" groups.
- The capabilities of carrying and exchanging interpreters provide a flexible way for a dynamic agent to be involved in multiple problem domains, even simultaneously.

Supporting the switch of problem domains and handling dynamic ontology, are the key mechanisms we use to approach the dynamic requirements of E-Commerce.

4. MULTI-AGENT COOPERATION WITH PLUG-IN WORKFLOW SUPPORT

Workflow systems provide flow control for business process automation. Business processes often involve multilevel collaborative and transactional tasks. Each task represents a logical piece of work that contributes to a process. A task at leaf-level is performed by a *role*. A role is filled at run-time with a user or a program. A process and its tasks are handled at separate layers. At the process layer, centralized coordination is supported; and at the task layer, location distribution, platform heterogeneity and control autonomy, are allowed [7,9,10].

Business processes may be considered as a kind of multi-agent cooperation, in the sense that a software agent can be used to fill a role for performing a task in a workflow, and workflow can be used to orchestrate or control the interactions between agents. However, many related activities in E-Commerce automation do not form synchronized, traditional workflow, but requires more dynamic agent cooperation. In order to combine the strength of workflow and agent cooperation for supporting E-Commerce, it is necessary to understand their relationship and difference.

First, workflow supports task integration with pre-defined flow control. Although a process could be modified through executing event-driven rules (e.g. ECA rules [9]), through enforcing inter-activity dependencies (e.g. do not book hotel if flight reservation is not made [7,10]), or through failure recovery [6], those alterations are also pre-defined. In contrast, agent cooperation is more dynamic and flexible. Focusing on a general goal, the task performed by agents may be dynamically selected, depending on run-time situation such as the results of the previous tasks. For example, choosing a purchase task depends on the results of negotiation with multiple seller agents, and the negotiation itself is a multi-agent cooperative, asynchronous process.

Next, the role of a software agent played in E-Commerce automation should be closer to that played by a human user (rather than a program) in the workflow context. In conventional workflow, a program task has a designated execution life span, it exists only during the execution time, and cannot receive messages before and after that task. On the contrary, a human user has memory and knowledge, and can work across multiple tasks and multiple business processes, even simultaneously. These properties are required for cooperative software agents. As a simple example, a buyer agent may simultaneously participate in several business partnerships with different vendors and brokers.

Our conclusion is that agent cooperation and workflow cannot replace each other, but may plug-into each other.

4.1 Workflow Tasks Executed by Multi-agent Cooperation

By plugging multi-agent cooperation in a workflow (Figure 5), we mean that a particular part of the workflow, such as a single task, may be accomplished by multiple agents working cooperatively. As an example, a purchase task may include bargain search and negotiation involving multiple agents. Such activities are handled by autonomous agents rather than under centralized workflow control. As another example, task reallocation among self-interested agents aimed at balancing work load, is also not centrally controlled. These tasks are performed through multi-agent cooperation.

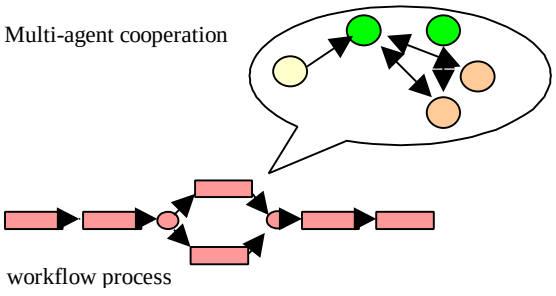

Figure 5: Plug Multi-Agent Cooperation in Workflow

Our dynamic agent infrastructure is suitable for plugging agent cooperation in workflow. This is because a dynamic agent is not simply a task, but a ***carrier*** of tasks that represents steps of a business process. Compared with a normal task, a dynamic agent is a continuous running object with persistent communication channel and knowledge across tasks and processes. In these aspects a dynamic agent can behave in a way more similar to a human user than to a normal program task. This allows, for example, an auction agent to use the above capabilities to combine requests from multiple buyers, and to make intelligent decisions by cooperating with other agents. Considering "selling items by auction" as a single task, it actually involves multi-agent cooperation.

4.2 Multi-agent Cooperation using Workflow Services

Multi-agent cooperation is more general than workflow. However, in some case there exists a need for workflow support, in order to *synchronize* agent cooperation[15]. Particularly the following transactional workflow features may be required:

- transaction properties known as ACID (atomicity, concurrency, isolation and durability) over a series of tasks;
- failure recovery [6,7,9,10] for multiple agents to roll back, totally or partially, a business process; and
- separation of process definitions and tasks.

Figure 6 shows how a business process may be plugged in agent cooperation, as shown. By plugged-in we mean two things. First, a *workflow process* is **launched on the fly** at a certain phase of agent cooperation to provide transaction

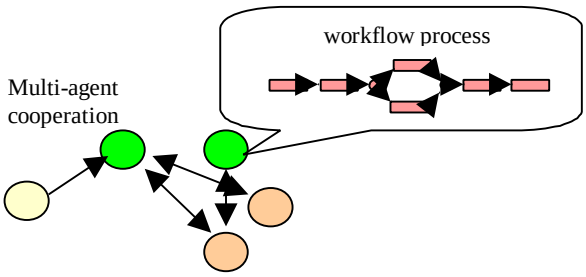

Figure 6: Plug Workflow in Multi-Agent Cooperation

properties, recovery, etc; second, a *workflow service*, comprising necessary engines, is constructed on the fly.

The use of dynamic agents makes it flexible to have workflow plugged-in agent cooperation. A process definition (as an object) can be loaded to a dynamic agent at run-time, and a process instance can be dynamically created and executed. Workflow engines (also as objects) may be downloaded, configured and setup on the fly. As described next

4.3 Dynamic Workflow Service Provisioning

The most promising feature of plugging workflow in agent cooperation is not launching a process to be executed by a

stand-by workflow server, but establishing workflow service on the fly.

In statically structured distributed systems, service is provided by stationary servers. The introduction of dynamic-agents can liberate service provisioning from such a static configuration. Given the underlying support for communication, program-flow, action-initiation, and persistent object storage, dynamic-agents can be used as the "nuts and bolts" to integrate system components on the fly, and thus to provide services dynamically. Consider the following scenario.

- Some agents watch inventory and sales trend.
- Other agents watch supply chain changes.
- When an agent, say ***A***, that correlates information from multiple sources, discovers an inventory shortage, it configures a purchase process, *order_proc* to order new products. This job includes two general steps: ***A*** first downloads and sets-up a workflow engine on the fly, and then sends it *order_proc* for execution. This process involves multiple tasks on remote sites, performed by agents as well as human users.

The workflow engine is made of two servers, a Process Manager for the flow-control of tasks, and a Work Manager for task dispatching and undispatching. These services are created on the fly in several message enabled steps described below.

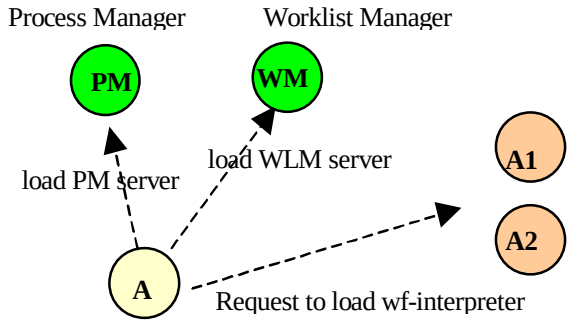

Figure 7: Launch Agents, Load Servers and Domain-specific Message Interpreters

As shown in Figure 7, ***A*** first launches dynamic-agents ***PM***, ***WM*** for loading the above workflow servers, as well as dynamic-agents ***A1*** and ***A2*** for carrying program tasks later. Then ***A*** sends messages to ***PM***, requesting it to download server Process Manager; and to ***WM***, requesting it to download server Work Manager. In this way, the workflow service is created on the fly. Next, ***A*** sends messages to ***PM***, ***WM***, ***A1*** and ***A2***, requesting each of them to download a workflow-message interpreter, for them to understand the work-items that will be assigned to them by ***WM***.

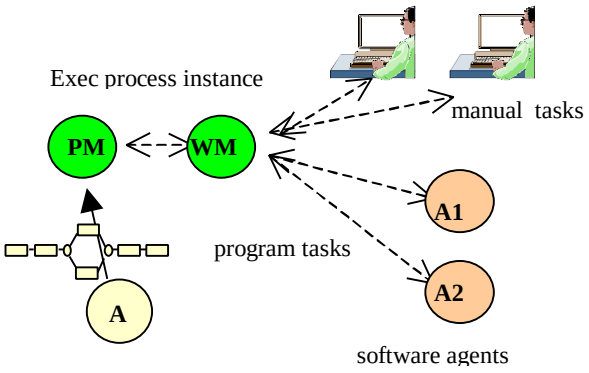

Figure 8: Send Process to Workflow Servers to Execute

After that, ***A*** starts process *order_proc* by sending a message to itself, or by an API call. A process instance instantiated at ***A*** is then sent to the Process Manager executing on ***PM***. This is illustrated in Figure 8.

The contents of the above messages are listed in Figure 9.

Tasks are then sent to the Work Manager executing on ***WM*** in order. Work items are generated by the Work Manager where manual tasks are sent to users (via a Web-browser), program tasks are sent to dynamic-agents ***A1*** and ***A2***, requesting them to download task-oriented programs first and then execute them; execution results will be sent back to Process Manager for flow control.

Fr	To	Message Content
A	A	<LAUNCH> <AGENT name="PM" host="host1.hp.com"/> <AGENT name="WM" host="host1.hp.com"/> <AGENT name="A1" host="host1.hp.com"/> <AGENT name="A2" host="host1.hp.com"/> </LAUNCH>
A	PM	<ACTION class="da.ProcMgr" url="file:host.hp.com/Wfclasses"> ProcMgr </ACTION>
A	WM	<ACTION class="da.WorkMgr" url="file:host.hp.com/Wfclasses"> WorkMgr </ACTION>
A	PM WM A1 A2	<LOAD_INTERPRETER class="da.WfInterpreter" url="file:host.hp.com/Dmclasses"> wf </LOAD_INTERPRETER>
A	A	<ACTION class="orde_proc" url="file:host.hp.com/Wfclasses"> order1 </ACTION>

Figure 9: Message Contents

A message containing a work item has the form:

```
<MESSAGE type="ASSIGN" from="WM" to="A1"
 interpreter="wf">
   <CONTENT>
     <TASK><ITEM> task_string </ITEM></TASK>
   </CONTENT>
</MESSAGE>
```

After the process is completed, the dynamic workflow servers may be shutdown, or even removed from the carrying dynamic agents. This example illustrates the power of dynamic-agents for plugging workflow in agent cooperation.

5. CONCLUSIONS

E-Commerce is a dynamic, distributed and a plug and play environment for which we expect software agent based technologies to become increasingly important. However, since agents with static capability cannot dynamically load new

functions, cannot change their predefined behavior, and cannot exchange programs with others, they are unable to switch roles, to participate in multiple applications, or to be involved in dynamically formed partnerships. Therefore, static agent frameworks are not really suitable for the highly dynamic E-Commerce applications.

In this paper we presented our solutions for E-Commerce automation using a dynamic agent infrastructure. Dynamic agents are carriers of application programs, they can be loaded with new functions, change behavior dynamically, and exchange programming objects. As a result, dynamic agents can switch roles, participate in multiple problem solving domains, and form dynamic partnerships. With these features, we have developed the mechanisms for plugging workflow and multi-agent cooperation in each other. In particular, we support dynamic workflow service provisioning, allowing workflow servers to be built on the fly. XML is chosen as our agent communication message format. Since different problem domains have different underlying ontology, we allow agents to communicate with domain specific languages (all in XML format) and act using corresponding interpreters. The program carrying capability of dynamic agents allows them to carry, switch and exchange interpreters. This not only enables dynamic agents to communicate using XML encoded messages with meaningful structure and semantics, but also to exchange domain ontology and message interpreters.

We plan to further explore a conceptual business model of plug and play E-Commerce.

6. REFERENCES

[1] Aglets, "Programming Mobile Agents in Java", IBM, http://www.trl.ibm.co.jp/aglets/, 1997.

[2] T. Bray, J. Paoli, C. M. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0 Specification", February 1998, (http://www.w3.org/TR/REC-xml)

[3] A. Chavez and P. Maes, Kasbah: An Agent Marketplace for Buying and Selling Goods, Proc. First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, 1996.

[4] Q. Chen, P. Chundi, Umesh Dayal, M. Hsu, "Dynamic-Agents", International Journal on Cooperative Information Systems, 1999, USA.

[5] Q. Chen, P. Chundi, U. Dayal, M. Hsu, "Dynamic-Agents for Dynamic Service Provision", Proc. of 3rd Int. Conf. on Cooperative Information Systems (CoopIS'98), 1998, USA.

[6] Q. Chen and Umesh Dayal, "Failure Recovery across Transaction Hierarchies", Proc. of 13th International Conference on Data Engineering (ICDE-97), 1997, UK.

[7] Q. Chen and Umesh Dayal, "A Transactional Nested Process Management System", Proc. of 12th International Conference on Data Engineering (ICDE-96), 1996, USA.

[8] CORBA, "CORBA Facilities Architecture Specification", OMG Doc 97-06-15, 1997.

[9] U. Dayal and M. Hsu and R. Ladin, "Organizing Long Running Activities with Triggers and Transactions", Proc. ACM-SIGMOD'90, 1990.

[10] U. Dayal and M. Hsu and R. Ladin, "A Transactional Model for long Running Activities", Proc. VLDB'91, 1991.

[11] Document Object Model, http://www.w3.org/DOM/

[12] T. Finin, R. Fritzson, D. McKay, R. McEntire, "KQML as an Agent-Communication Language", Proc. CIKM'94, 1994.

[13] Foundation for Intelligent Physical Agents(FIPA)- FIPA97 Agent Specification, http://www.fipa.org/

[14] R. J. Glushko, J. M. Tenenbaum and B. Meltzer, "An XML Framework for Agent-based E-Commerce", CACM 42(8), March, 1999.

[15] M. Griss, G. Bolcer, L. Osterweil, Q. Chen, R. Kessler "Agents and Workflow -- An Intimate Connection, or Just Friends?", Panel, TOOLS99 USA, Aug 1999.

[16] R. S. Gray. Agent Tcl: A flexible and secure mobile-agent system. Dr. Dobbs Journal, 22(3):18-27, 1997.

[17] N.R. Jennings, P. Faratin, MJ. Johnson, P O'Brien & ME Wiegan, "Using Intelligent Agents to Manage Business Processes". Proc. of PAAM96, U.K., 1996, pp. 245-360.

[18] N. R. Jennings (1999) "Agent-based Computing: Promise and Perils" Proc. IJCAI-99, Sweeden. 1429-1436.

[19] T. John, Intelligent Agent Library/Factory, release 4 (http://www.bitpix.com)

[20] P. Maes, R. H. Guttman and A. G. Moukas, "Agents that Buy and Sell", CACM 42(8), March, 1999.

[21] S.A. Moore, "KQML and FLBC: Contrasting Agent Communication Languages," proceedings. 32nd Hawaii international conference on system sciences, 1998,

[22] A.G. Moukas, R. H. Guttman and P. Maes, "Agent Mediated Electronic Commerce: An MIT Media Laboratory Perspective", Proc. of International Conference on Electronic Commerce, 1998.

[23] Odyssy, "Agent Technology: Odyssey", General Magic, http://www.genmagic.com, 1997.

[24] B. Perry, M. Talor, A. Unruh, "Information Aggregation and Agent Interaction Patterns in InfoSleuth", Proc. of CoopIS'99, UK, 1999.

[25] Voyager, "Voyager Core Package Technical Overview", Object Space, http://www.objectspace.com/voyager/technical_white_papers.html, 1997.