Dynamic-Agents, Workflow and XML for E-Commerce Automation

Qiming Chen, Umesh Dayal, Meichun Hsu, Martin Griss

HP Labs, Hewlett-Packard, 1501 Page Mill Road, MS 1U4, Palo Alto, CA 94303, USA
{qchen,dayal,mhsu,griss}@hpl.hp.com

Abstract. Agent technologies are now being considered for automating tasks in e-commerce applications. However, conventional software 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 dynamic service provisioning, allowing services to be constructed on the fly. We treat an agent as a Web object with an associated URL, which makes it possible to monitor and interact with the agent remotely via any Web browser. XML is chosen as our agent communication message format. Dynamic agents can carry, switch and exchange domain-specific XML interpreters. 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.

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.

E-Commerce applications operate in a *dynamic and distributed environment*. An E-Commerce scenario typically involves the following activities: identifying requirements, brokering products, brokering vendors, negotiating deals, and making purchase and payment transactions. Today, these activities are initiated and executed primarily by humans. In the future, with the increasing automation of e-commerce, we see them being conducted by software agents (Figure 1).

Software agents are personalized, continuously running and semi-autonomous computational entities. They can be used to *mediate* users and servers to automate a number of the most time-consuming tasks in E-Commerce [7,8]. Agents can also be used for *business intelligence*, such as discovering patterns (e.g. shopping behavior patterns or service providing patterns) and reacting to pattern changes. Moreover,

agents can selectively preserve data and themselves become *dynamic information sources*.

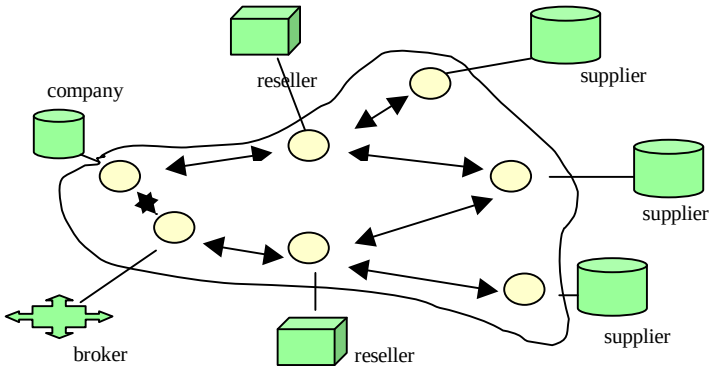

Figure 1: E-Commerce automation through multi-agent cooperation

E-Commerce is also a *dynamic 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 such dynamic business partnerships. Furthermore, agents may cooperate in different application domains. The dynamic nature of E-Commerce also requires multi-agent cooperation to be based on *dynamic ontology*.

At HP Labs, we have developed a Java based *dynamic agent* infrastructure for E-Commerce that supports *dynamic behavior modification* of agents, a significant difference from other agent platforms [1]. 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.

Dynamic agents are designed as Web objects. Each agent contains or connects to a Web server, and has a dedicated Web page, accessed via a URL, on which is posted information about the agent's state and the tasks it is carrying.

Dynamic agents communicate using XML[5], and can *dynamically load and exchange different ontologies and XML interpreters* for tasks in different domains.

Finally, we introduce mechanisms for *modeling and enacting agent cooperation* as *workflow processes*, and *plugging in cooperating agents to execute tasks* of workflow processes. In particular, *dynamic 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, unified mechanisms for data flow, control flow, and even program flow among agents for supporting E-Commerce.

These approaches also differentiate our agent infrastructure from those that lack dynamic modifiability of behavior [9,11] in the sense that an agent must be statically coded and launched for doing only a fixed set of things. Our approach to providing workflow services on the fly during multi-agent cooperation, is also unique. We share the same view as [2,5] on the importance of XML for E-Commerce. In fact, several agent communication languages such as FIPA[4], KQML[3], etc, have been converted to simple XML form. However, our dynamic agents can exchange domain specific ontology, and even XML interpreters, so as 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 shows how to build agents as Web objects. Section 4 describes the use of XML messaging with dynamic ontology to support multi-agent cooperation. Section 5 illustrates how to plug workflow processes into multi-agent cooperation, and vice versa. Finally in section 6, some concluding remarks are given.

2 Dynamic Agents

To realize E-Commerce automation, agents need to have dynamic behavior while maintaining their 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 [1]. 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.

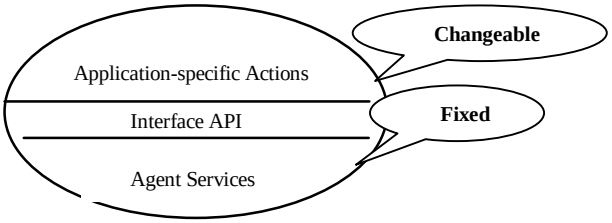

Figure 2: Dynamic Agent as Program Carrier

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, knowledg, 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, application-specific programs.

The architecture of a dynamic agent is described in more detail below.

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, for 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 instantiate and execute 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; it 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 these 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 [4], a coordinator agent is used to provide a naming service, mapping the name of each agent to its current socket address. The coordinator itself 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 dynamic agent 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 a naming service, other types of coordination may be required, and these are provided either by the coordinator or by other designated dynamic agents that provide brokering services for resources, requests and events.

Dynamic agents form a dynamic distributed computing environment for E-Commerce. In the following sections we shall show how to take advantage of the dynamic agent infrastructure for E-Commerce automation.

3 Agents as Web Objects

The reach of the World Wide Web (WWW) has extended out from all varieties of computing platforms into the domain of small appliances. Web-enabled appliances can be monitored or interacted with remotely. We see this trend moving from hardware boxes to software modules. A dynamic agent with an embedded web server can deliver a web page to a browser on a remote computer, and this page can be used to interact with the agent and to monitor or manage the agent remotely. The advantage of doing so goes far beyond providing a more flexible GUI.

The main difference between treating agents as Web objects and as other Java objects is that each agent has a Web page, which is accessed via a URL. This Web page contains information about the agent itself and about the task it is doing. With this mechanism we can build a virtual market where clients and servers are all connected to the Web, have Web-facing representations, and are able to offer and participate in services on the Web.

As a Web object, an agent connected to the Internet is accessible via HTTP or XML. This typically requires that the agent embeds, or accesses, a web server. An agent is provided with at least one web page (HTML or XML page) that provides an interface for it to be accessed via any browser. The agent "publishes" on that page, a set of control, maintenance and application operations that can be accessed or invoked via Web browsers.

For example, an agent carrying seller functionality has a web page that shows what it sold, for which price, to whom, and when. The control operations can be represented on the Web page. An authorized person can adjust the price range via the Web page.

As with the agents' names, the URLs of dynamic agents are maintained by the coordinator as a kind of resource in case the agents move, but the coordinator's URL (host + port) is left stable.

An operation request issued from a Web browser to a dynamic agent looks the same as a traditional form request. For example, operation requests may be sent over HTTP using the GET or POST method. In the case of a POST request, the operation

name and argument data will be sent in the body of the request. In a standard Web request, the arguments in the URL (including the operation name and return value) will be URL-encoded, which is a simple text encoding scheme. More complex encoding schemes can be used to support integration with non-Web based systems or for the passing of private information types. Note that inter-agent communication using messages does not require the involvement of a Web server. Interaction with an agent via a Web browser can easily interface to agent communication messaging facilities. For example, such a facility may be configured as a programmatic gateway supported on a Web server, which is easy to do by specifying the corresponding MIME table.

4 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, are key issues. The message format should be accepted not only by the agent research community, but should be an industry standard that is likely to be adopted by information providers.

4.1 Document-driven Agent Cooperation

The extensive markup language, XML, is fast becoming the standard for data interchange on the Web. We chose XML, therefore, as the primary message format for dynamic agent communication.

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 the Web accessible to agents with XML, the need for custom interfaces for each consumer and supplier will be eliminated. Agents may use an XML format to explain their BDI, defining new performatives in terms of 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 the ontology. As ontology varies from domain to domain, and may even be dynamic for dynamically formed domains. A significant issue is how to exchange the semantics of domain models, and how to interpret messages differently in different problem domains.

Generally speaking, a domain ontology provides a set of concepts, or meta-data, that can be queried, advertised and used to control the behavior of cooperating

agents. 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 the DTDs together with documents, and exchange the corresponding interpreters as programming objects, in order to understand each other.

4.2 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 sequence. 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 an 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 [2]).

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.

4.3 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>
```

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

5 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. Business processes may be considered as a kind of multi-agent cooperation [6], 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 require 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 differences.

First, workflow supports task integration with pre-defined flow control. In contrast, agent cooperation is more dynamic, flexible, and decentralized. Focusing on a general goal, the task performed by agents may be dynamically selected, depending on the run-time situation such as the results of previous tasks. For example, choosing a purchase task depends on the results of negotiation with multiple seller agents, and the negotiation itself is a cooperative, asynchronous multi-agent 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.

Plugging Multi-agent Cooperation into a Workflow. By this we mean that a particular part of the workflow process, 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 workload, is also not centrally controlled. These tasks are performed through multi-agent cooperation.

Our dynamic agent infrastructure is suitable for plugging agent cooperation into 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 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. From the workflow point of view, "selling items by auction" may be considered a single task, but it actually involves multi-agent cooperation.

Plugging in Workflow to Multi-agent Cooperation. Multi-agent cooperation is more general than workflow. However, in some cases there exists a need for workflow support, in order to synchronize agent cooperation[6]. Particularly, the transactional workflow features such as ACID and failure recovery, may be required.

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 services 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 trends. 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. After the process is completed, the dynamic workflow servers may be shut down, or even removed from the carrying dynamic agents.

6 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, agents with static capability 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 program 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 mechanisms for plugging workflow and multi-agent cooperation into 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.

A very practical and significant development is to build our dynamic agents as Web objects, which allows them to be monitored, accessed and controlled remotely in a unified way. This mechanism represents two steps towards the virtual marketplace, mediating physical resources with agents, and mediating agents via the Web.

References

1. Q. Chen, P. Chundi, Umesh Dayal, M. Hsu, "Dynamic Agents", International Journal on Cooperative Information Systems, 1999, USA.
2. Document Object Model, http://www.w3.org/DOM/
3. T. Finin, R. Fritzson, D. McKay, R. McEntire, "KQML as an Agent-Communication Language", Proc. CIKM'94, 1994.
4. Foundation for Intelligent Physical Agents(FIPA)- FIPA97 Agent Specification, http://www.fipa.org/
5. R. J. Glushko, J. M. Tenenbaum and B. Meltzer, "An XML Framework for Agent-based E-Commerce", CACM 42(8), March, 1999.
6. M. Griss, G. Bolcer, L. Osterweil, Q. Chen, R. Kessler "Agents and Workflow -- An Intimate Connection, or Just Friends?", Panel, TOOLS99 USA, Aug 1999.
7. P. Maes, R. H. Guttman and A. G. Moukas, "Agents that Buy and Sell", CACM 42(8), March, 1999.
8. A.G. Moukas, R. H. Guttman and P. Maes, "Agent Mediated Electronic Commerce: An MIT Media Laboratory Perspective", Proc. of Int. Conf. on Electronic Commerce, 1998.
9. Odyssy, "Agent Technology: Odyssey", General Magic, http://www.genmagic.com, 1997.
10. B.Perry, M. Talor, A. Unruh, "Information Aggregation and Agent Interaction Patterns in InfoSleuth", Proc. of CoopIS'99, UK, 1999.
11. Voyager, "Voyager Core Package Technical Overview", Object Space, http://www.objectspace.com/voyager/technical_white_papers.html, 1997.