A Multi-agent Framework for Intranet Service Integration

Jane Yung-jen Hsu

Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 106
yjhsu@csie.ntu.edu.tw

Abstract. Networked computing has drastically changed the way in which people work and exchange information. Although the standard client-server architecture enables sharing of common resources among multiple users, it is non-trivial to share computational resources over the network. In this paper, we present a *client-agent-server* framework for integrating intranet services in satisfying complex informational tasks. Agents play the key role in linking individual commands into useful plans, and in transforming heterogeneous data into valuable knowledge. The agent-based approach reduces the burden of using distributed services for the end users, and fills the gap between information service providers and clients. In particular, a community of printer agents are presented to demonstrate the advantages of the proposed multi-agent framework.

1 Introduction

The world-wide web has become the most effective means for people to share *data*, i.e. hyper-linked documents, over the Internet. Meanwhile, an intranet, which refers to the application of Internet standards and systems to the management of internal networks, enables people to work and communicate efficiently within an organization. While many computational *services* are available over the intranet, it is non-trivial to integrate and share such resources.

Consider a network consisting of multiple (possibly heterogeneous) servers interconnected with multiple clients. The former offer a variety of data and computational services, while the latter issue a variety of informational requests. Under the standard client-server architecture, utilizing shared server resources often requires sophisticated user manipulation or even programming. For example, suppose that John wants to produce a printed copy of a report, which appears as an attachment `report.tar.gz` to an email from his colleague. First of all, he opens an email application, finds the email message, and saves the attached file to a local hard disk. He then has to unzip and untar the file, only to find that the report is in fact contained in multiple `.tex` and `.ps` files. Undaunted by the fact that `latex` is not readily available on his machine, John proceeds to transfer the files to another machine, processes the files, sets up the printer, and finally generates a printed document. The seemingly simple task

of “printing a document” can be laborious for expert users, and challenging or mission-impossible for novices, especially when the utilities and applications are scattered on different machines within a distributed environment.

As is illustrated in the scenario above, there are several problems with how computers are used today.

- Computers and software applications are viewed as tools to be manipulated directly by the users.
- In order to solve a single task, a sequence of software utilities and applications may need to be deployed manually.
- There is no easy way to instruct the computers at an intuitive level. The current generation of graphical user interface are friendly looking, but the mode of interaction is passive in nature.
- With increased functionalities, software are growing in terms of both program size and resource requirement.
- The applications can’t communicate to coordinate their functions or data transfer automatically.

In recent years, intelligent agents have emerged as a new software paradigm in which users are able to *delegate* tasks to the computers. There is a wide variety of agents [1]. While the definition of an agent is still a subject of much controversy, we take an agent to be “any program that can be considered by the user to be acting as an assistant or helper, rather than as a tool in the manner of a conventional direct-manipulation interface” [15]. For example, the SoftBots [8,7] allow a user to specify (incompletely) a task, e.g. send emails to someone, and the computer then carries out the appropriate sequence of commands in order to accomplish the task. For the softbots to work properly, all system utilities are assumed to be at their disposal. Additionally, multi-agent systems offer a modular solution for domains that are particularly complex, large, or unpredictable [18].

Instead of viewing the computer as a desktop with an array of tools, the user should be able to interact with the computer by delegating tasks to the team of agents who manage the intranet services for her. This research attempts to establish a multi-agent framework for the problem of *intranet services integration*. Section 2 starts by formulating the problem. A three-tier *Client-Agent-Server* is introduced in Section 3, and a hierarchy of agents are defined in Section 4. Some important issues in designing such an infrastructure are also discussed. Section 5 presents a case study of AutoPrint agents that assist intranet users in printing documents. This paper concludes by outlining some future research directions.

2 The Service Integration Problem

Manipulating multiple services on distributed machines is problematic. Due to the mismatch between the level of services provided and the level of tasks requested, it takes a lot of skills on the user’s part to *coerce* the computer to do the right things. A user has to utilize the shared data and computational resources

effectively. This section formulates the problem of *service integration* in such a way that it can be decomposed into more manageable problems. Let us start by defining the basic components.

Definition 1. A service $s = \langle c, h, r, f \rangle$ is any computation that

- is invoked by a single command c ,
- can be executed by a single host h ,
- requires resource r , and
- achieves functionality f .

The first component defines the corresponding command for the service, and the other three components define its operating context.

A service can be any specific command, utility, or application available for execution on a host machine. A service functionality can be defined in terms of changes in the task domain. Services may be similar in their functionalities. For example, all print servers offer the same kind of service, e.g. increasing the number of hardcopies of a given file in the world, while each “`lpr -Ppname`” command on a specific host machine is considered a distinct service.

Definition 2. A service operator $S = \langle N, F \rangle$ is a set of services, where N is a unique service name and F is a distinct service function, such that a service s is in S if and only if $f(s)$ is an instantiation of $F(S)$.

In a sense, a service is an instantiation of a service operator with the proper operating context, e.g. the host information or resource requirements. Services of a given service operator produce similar state transitions in the problem space. The service operators may be further abstracted into *service operator schemas* by introducing variables in the same spirit of *operator schemas* in standard AI planning. Multiple service operators may be combined into a service script for task achievement.

Definition 3. Given a set Σ of service operators, a service script σ is a partially-ordered multi-set over Σ .

Consider a simplified version of the printing task discussed in the previous section. Given a file `f.tar` containing two gzipped files `f1.ps.gz` and `f2.tex.gz`, together with a $\text{\LaTeX}$ document `f.tex`. The document can be printed out using the service script shown in Figure 1. Each node in the script represents a service operator, and an operator o_1 has to be executed before another operator o_2 if there is an arc leading from o_1 into o_2 . The shaded operators are optional.

An *instantiated service script* is one in which each service operator has been instantiated by a specific service. Not all service scripts are executable given the current system configuration.

Definition 4. A service script is sound with respect to the current system configuration if there exists at least one instantiated service script such that the operating context for every service in the script is supported. That is, for each service $\langle c, h, r, f \rangle$ in the instantiated service script, the following conditions hold when all the preceding services have been completed.

Fig. 1. A Sample Service Script

1. Host h offers command c .
2. Resource r is available.

We are now ready to define the service integration problem and its solution as follows.

Definition 5. *Given a task specified in terms of a goal G , a set Σ of service operators, a collection of hosts H , each of which offers a subset of services from Σ , and a collection of shared resources R . The solution to the service integration problem $\langle G, \Sigma, H, R \rangle$ is to find a service script σ such that the following conditions are satisfied.*

1. The service script σ is sound.
2. Execution of the service script results in goal achievement.
3. (Optional) σ requires a minimal amount of resources among all scripts satisfying the first two conditions.

The benefits of an intranet depend on the effectiveness of service integration that can be supported. While it is indeed possible to program suitable service scripts for specific tasks, it is desirable for the job to be automated. In what follows, a multi-agent framework is proposed as a solution for automatic service integration.

3 The Client-Agent-Server Model

The client-server architecture has provided the cornerstone for distributed computing. In the model, a client program requests a service from a server program by taking the following steps:

1. Create a socket connection with the server.
2. Send a service-request message to the server.
3. Wait for a response from the server.

The server response may indicate further information needed, requested action performed, or provide the data requested. Communications between the server and the client depend on the specific protocol used by the application. It is the responsibility of the programmer to ensure that the types of services provided match the types of requests. Unfortunately, there is a glaring gap between

complex informational requests and services provided by individual application servers. A complex client request often has to be satisfied by linking services provided by multiple servers. For example, the client request of “print `f.tar`” cannot be satisfied by any single server offering `lpr` alone. Furthermore, a client may not have the information about which server(s) can offer the desired functionality, thus making Step 1 above virtually impossible.

In the client-agent-server model, multiple agents work to bridge the gap due to the mismatch between client requests and server services. The role of agents is two-fold. On the one hand, agents help integrate computational services in order to satisfy the goals of complex tasks. On the other hand, agents help integrate data in order to generate useful information. Figure 2 shows the overall architecture of the proposed three-tier model. Before we describe in detail the

Fig. 2. The Client-Agent-Server Model

functionalities of the various agents, let us examine the information flow in this architecture. A client request is handled by an interface agent, who breaks down the problem into *independent* pieces for a team of task agents. Each task agent computes an appropriate service script for solving the specific sub-problem. The generated script is then handed over to a service agent, who allocates the appropriate resources to carry out the given script. As a result, a user is able to delegate tasks rather than manipulating the services directly. This framework enables the user to utilize with ease any of the networked data or computational services in accomplishing a specific task. Furthermore, the services fulfilling the user tasks can be tailored toward his/her individual needs.

Similar ideas have been explored in the domains of integrating concurrent engineering systems [6], and integrating multiple databases [19]. Client requests are matched up with information sources with the help of a facilitator or broker. Bradshaw et al. [2] proposed a generic agent architecture KAoS that supports reuse, interoperability, and extensibility. Cao and Shi also proposed a Common

Agent Request Broker Architecture (CARBA) [3]. Both KAOs and CARBA follow the ideas from the CORBA distributed object architecture. The proposed model is unique in the following ways:

- The framework supports integration of *computational services* in addition to data from multiple servers.
- It avoids the bottleneck of mediating agents using a hierarchical approach, which will be described in more detail in the next section.

4 A Hierarchy of Agents

The community of agents in the proposed client-agent-server model should be designed with care. It is relatively easy to become another *sweep-under-the-rug* model by pushing the hard problems to an “unimportant” black box. This section starts by examining the different styles of software design, defines a hierarchy of abstraction spaces for different agents, and compares their features. The hierarchy of agents are described in Section 4.2.

4.1 Agent-Oriented Software Design

Agent-orientation is becoming a new paradigm in software design. While object-oriented programming has been dominant for the past decade, objects are *passive* in the sense that their actions are initiated by external stimulation, e.g. messages from other objects. In contrast, agents are active objects that can function autonomously and/or pro-actively. Designing software today follows a dramatically different model from designing monolithic computer programs. Several progressively more complex design models are introduced below.

The Soloist Model. In the age of writing monolithic programs, the programmer is responsible for defining all the details of the desired computation. A skilled programmer is like a soloist who knows his instrument and the effects of performing any specific action on it. The programmer is able to carry out complicated computations in the same manner that a soloist is able to perform difficult passages of music.

The Conductor Model. With multiple heterogeneous computers available, the programmer has to understand the features of each in order to coordinate the data or computation properly. A skilled programmer is like a conductor who knows all the instruments and the effects of performing actions on them. The programmer is able to solve complex tasks in the same manner that a conductor is able to direct a symphony orchestra performing a concert. There is little wonder that good programmers for distributed applications are hard to find.

The Manager Model. To manage the increased complexity in large applications, the concept of agents has been introduced. A good manager should delegate tasks, be responsible for setting up reasonable performance goals, understand the characteristics of his team members, and maintain their autonomy. Likewise, a skilled programmer needs to organize her team of agents based on their features and functionalities. A programmer is able to accomplish complex tasks in the same manner that a manager is able to lead his team in meeting project goals. Instead of having to interact with an arbitrary number of agents, a hierarchical organization is used in the proposed model.

The Market Model. For applications involving a large number of agents, but relatively little domain knowledge, the market model may be a good idea. The programmer is responsible for defining the basic architecture of individual agents as well as the protocols governing their interactions. The functionalities of each agent may be simple. Complex behavior of the multi-agent system emerges as dynamic interactions among its constituent agents.

4.2 Abstraction Layers for Multiple Agents

An important feature of the proposed multi-agent framework is its use of a hierarchy of abstraction spaces to manage complexity. The idea is based on *abstraction planning* [17] in which a solution to a goal is found in an abstract space, and the solution is used as a skeleton for a more detailed plan at a less abstract space. As is shown in Figure 3, an interface agent tries to solve a given task by identifying the appropriate team of task agents that should participate in the solution. Each task agent solves his problem in the space of service operators. A candidate service script is generated based on task-specific knowledge about the domain. The service agents attempt to instantiate the service script by allocating a service to each of its service operator. An advantage of this approach is that the interface (service) agents only need to work within the well-defined space of agents (services). Task agents may require varying levels of planning capabilities, depending on the specific task domains.

Interface Agents. Such an agent operates in the interface, as opposed to in the background or “back end” of an application. In our framework, each interface agent corresponds to a class of client requests, e.g. for a specific user or a specific task. The basic functionalities include:

- Accepting task specification from the user,
- Organizing the team of agents for the specified task,
- Presenting results in response to the task request,
- Logging personal information and user behavior,
- Conducting conversations with the user, and
- Offering user help pro-actively (at the right time).

Fig. 3. Three Layers of Abstraction

Task Agents. Task agents act autonomously, as opposed to having a sequential conversation with the user. Their basic functionalities include:

- Maintaining task-specific domain knowledge,
- Accepting goal specification from the interface agents,
- Generating service script w.r.t. the goal specification, and
- Replanning on demand.

Service Agents. Service agents are also called *resource* agents. Each service agent corresponds to a specific server host. Their basic functionalities include:

- Monitoring and managing local system resources, e.g. application software or peripheral devices etc.,
- Accepting service script from the task agents,
- Executing service script by allocating appropriate services,
- Maintaining *service registry*, and
- Communicating with agent service registry.

In conclusion, the characteristics for the three classes of agents are summarized and contrasted in Table 1.

Table 1. Characteristics of Agents

Characteristics	Interface Agents	Task Agents	Service Agents
autonomous	Maybe	Yes	Yes
pro-active	Yes	No	No
problem solving	team formation	planning	service allocation
operation space	agents	service operators	services
knowledge	user-oriented	task-oriented	machine-oriented
life span	persistent	episodic	episodic
communication	user+agent	agent	agent+administrator

4.3 Agent Infrastructure

The Huhns-Singh Test for Agenthood: A system containing one or more reputed agents should change substantively if another reputed agent is added to the system [12].

Many multi-agent architectures have been proposed in the past [5,4], but none of them provide the necessary support for agents operating at different levels of abstraction. To support the hierarchy of agents in this research, an object-oriented architecture called ARBIS (Agent Request Broker InfraStructure) is being developed. In particular, ARBIS is responsible for the following functions:

- Providing agent brokering/naming services,
- Creating and killing agent instances on demand,
- Sustaining life for active agents,
- Maintaining agent registry,
- Maintaining service operator registry,
- Managing user profiles, and
- Interfacing with the underlying communication protocols.

Just as CORBA is a standard for interoperability in heterogeneous computing environments [11,9], ARBIS intends to provide a standard for interoperability for agents. In the current framework, agents communicate with each other and with ARBIS using a simplified version of KQML [10]. The goal is that any ARBIS-compliant agent can be easily incorporated into an existing multi-agent application.

5 Case Study: AutoPrint Agents

We have experimented with the proposed multi-agent service framework in the domain of intranet printing[13,14]. Most of us have experienced problems such as printing out the source of a huge postscript file by accident, or fretting over the proper command sequence in preparing a document for printing. With the growth of multimedia applications, there is a greater proliferation of file formats in routine use. Consider a number of interconnected machines, each of which offers any number of services related to printing and file conversion. The experimental environment is as shown in Figure 4, and Table 2 summarizes the host configuration in terms of services registered at each machine.

The AutoPrint interface agent interacts with the users through dialog boxes such as the ones shown in Figure 5. One important principle in our design is that the interface agents should request minimal information from the user in order to simplify task specification. In general, the user only needs to interact with the agent via the dialog box on the left. The interface agent makes decisions about the default values for any unspecified fields as well as the advanced printing options based on statistics collected for each user. The (incompletely specified) print task is passed to AutoPrint task agent, which generates a service script using automatic planning provided by UCPOP [16]. The service scripts are reused when appropriate in an effort to reduce the overhead due to planning [13].

A sample mix of test files were created and the following experimental steps were repeated one thousand times.

1. Register a randomly selected subset of the services for each host.
2. Create a new print job request by randomly selecting a test file and a printer.

Table 3 summarizes the *average time* taken by script planning, service allocation, and service execution. The experiments showed that the AutoPrint agents are able to complete the print tasks successfully under a wide range of randomly

Fig. 4. Host Environment for AutoPrint Agents

Table 2. Host Configuration

(a) Hardware/Software Configuration

Machine	Hardware	OS	A	B	C	D	E	F	G	H
agent	Sparc 20	SunOS 4.1.4	✓	✓	✓	✓	✓	✓	✓	✓
nomad	Sparc 5	SunOS 4.1.4		✓	✓	✓	✓	✓	✓	✓
super	Sparc 4	Solaris 2.5.1	✓		✓	✓	✓	✓	✓	✓
ntucsl	Sparc 10	SunOS 4.1.3		✓				✓	✓	
ntucsn	Sparc 5	SunOS 4.1.3		✓				✓	✓	
ntucsv	UltraSparc 1	Solaris 2.5.1		✓	✓	✓	✓	✓	✓	

(b) Registered Services

Code	Service Operator	Function
A	print2hp3	print postscript file to printer hp3
B	print2hp5	print postscript file to printer hp5
C	tex2dvi	convert tex file into dvi file
D	dvi2ps	convert dvi file into ps file
E	pdf2ps	convert pdf file into ps file
F	uncompress	uncompress .Z files
G	gunzip	uncompress .gz files
H	ucpop	partially-ordered planner

(a) Task Specification

(b) Printing Options

Fig. 5. AutoPrint User Interface

Table 3. Experimental Results

File	Size (Bytes)	Planning	Allocation	Execution
test.tex	3214	29.494	3.799	15.041
test.tex.Z	1399	54.318	5.821	9.979
test.tex.gz	885	49.135	7.897	12.555
test.dvi	4004	24.776	0.951	8.033
test.dvi.Z	2339	40.017	4.470	5.267
test.dvi.gz	1633	38.142	3.657	6.417
test.ps	19847	26.951	0.641	0.621
test.ps.Z	11172	29.533	2.163	0.966
test.ps.gz	8385	33.484	3.798	1.084
test.pdf	7088	23.896	2.078	4.470
test.pdf.Z	8540	42.914	2.807	5.215
test.pdf.gz	5979	38.177	5.301	4.336

generated service configurations. A plan library can be used by the AutoPrint agents to reduce the planning time to about 10% of the original time. In addition, the agents are shown to be robust with respect to service break-downs [14].

6 Conclusion

This paper proposed a three-tier *client-agent-server* architecture to bridge the gap between informational requests and services within a distributed environment. The multi-agent framework provides a powerful and flexible way to integrate arbitrary intranet services. The service integration problem is solved in a hierarchical fashion, with interface, task, and service agents working at different abstraction levels. Agents with user, task, or service-specific competence

can be easily incorporated into the framework. The current implementation, which works under the Unix environment, demonstrates the viability of such a multi-agent approach. Work is underway to develop the *Agent Request Broker Infrastructure* for heterogeneous service platforms. The goal is a *plug-and-play* architecture for agents that supports multiple agents in the same manner that CORBA supports distributed objects. The interface problem is still pretty open at this point. Coordinating service scripts among multiple task agents calls for sophisticated inter-agent communication and negotiation schemes, especially if the agents cannot be assumed to be independent. This research has identified the need for personalized information services, and a *profile management* mechanism is being designed. For better cross-platform operation, we also plan to move from the current tcl/tk implementation to a browser-based user interface.

Acknowledgments

The author would like to thank Ming-Wei Jean and Bu-Jian Li for their implementations of the AutoPrint agents. Thanks also go to Toru Ishida, Soe-Tsyr Yuan, and Jieh Hsiang for their helpful comments. Discussions over the last two years among members of the agents project at National Taiwan University have helped shape the ideas presented in this paper. This research was supported in part by grants from the National Science Council of ROC under grant numbers NSC86-2745-E-002-007 and NSC87-2213-E-002-014, and in part by Tjing Ling Industrial Research Institute of National Taiwan University.

References

1. J. Bradshaw, editor. *Software Agents*. AAAI/MIT Press, Menlo Park, CA, 1996.
2. J. M. Bradshaw, S. Dutfield, P. Benoit, and J. D. Woolley. KAoS: Toward an industrial-strength generic agent architecture. In J.M. Bradshaw, editor, *Software Agents*. AAAI/MIT Press, Cambridge, MA, 1996.
3. H. Cao and Z. Shi. CARBA: Common agent request broker architecture. In T. Ishida, editor, *Proceedings of the 1998 Pacific Rim International Workshop on Multi-agents*, November 1998.
4. D. Chess, C. Grosz, B. Harrison, D. Levine, C. Parris, and G. Tsodik. Itinerant agents for mobile computing. In M.N. Huhns and M.P. Singh, editors, *Readings in Agents*, pages 267–282. Morgan Kaufmann, San Francisco, 1998.
5. P.R. Cohen, A. Cheyer, M. Wang, and S.C. Baeg. An open agent architecture. In M.N. Huhns and M.P. Singh, editors, *Readings in Agents*, pages 197–204. Morgan Kaufmann, San Francisco, 1998.
6. M.R. Cutkosky, R. Englemore, R.E. Fikes, M.R. Genesereth, T.R. Gruber, W.S. Mark, J.M. Tenenbaum, and J.C. Weber. PACT: An experiment in integrating concurrent engineering systems. In M.N. Huhns and M.P. Singh, editors, *Readings in Agents*, pages 46–55. Morgan Kaufmann, San Francisco, 1998.
7. O. Etzioni, H. M. Levy, R. B. Segal, and C. A. Thekkath. Os agents: Using ai techniques in the operating systems environment. Technical Report 93-04-04, University of Washington, Seattle, August 1994.

8. O. Etzioni and D. Weld. A softbot-based interface to the internet. *CACM*, 37(7):72–76, July 1994. <http://www.cs.washington.edu/research/softbots>.
9. E. Evans and D. Rogers. Using java applets and corba for multi-user distributed applications. *IEEE Internet Computing*, 1(3):43–55, May-June 1997.
10. T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *The Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*. ACM Press, 1994.
11. Object Management Group. The common object request broker: Architecture and specification, revision 2.2. OMG 98-07-01, Object Management Group, February 1998. Also available at <http://www.omg.org/corba/cortiiop.htm>.
12. M.N. Huhns and M.R. Singh. Agents on the web. *IEEE Internet Computing*, 1(5):78–79, September/October 1997.
13. M.W. Jean. Intelligent printer agents for intranet environments. Master's thesis, National Taiwan University, Computer Science and Information Engineering, June 1997.
14. B.J. Li. Task agents and service sharing in local area networks. Master's thesis, National Taiwan University, Computer Science and Information Engineering, June 1998.
15. H. Lieberman. Autonomous interface agents. In *CHI'97*. ACM, 1997. <http://www.acm.org/sigchi/chi97/proceedings/paper/h1.html>.
16. J. S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for adl. In *Proceedings of KR-92*, pages 103–114, October 1992.
17. E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.
18. K. P. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, 1998.
19. G. Wiederhold. Mediators in the architecture of future information systems. In M.N. Huhns and M.P. Singh, editors, *Readings in Agents*, pages 185–196. Morgan Kaufmann, San Francisco, 1998.