ATHABASCA UNIVERSITY

AGENTS AS WEB SERVICES

BY

YLBER RAMADANI

A thesis essay submitted in partial fulfillment

Of the requirements for the degree of

MASTER OF SCIENCE in INFORMATION SYSTEMS

Athabasca, Alberta

January, 2005

ATHABASCA UNIVERSITY

The undersigned certify that they have read and recommend for acceptance the thesis project "AGENTS AS WEB SERVICES" submitted by YLBER RAMADANI in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE in INFORMATION SYSTEMS.

[illegible]

Larbi Esmahi, Ph.D.

Supervisor

[illegible]

XiaoKun Zhang, Ph.D., Chair

[illegible]

Harris Wang, Ph.D., Examiner

Date: April 06. 2005

DEDICATION

I dedicate this essay to my lovely wife Alma, my son Keltrid, and my newly born daughter Kaitlin. They have all supported me in their own ways. Without them it would have been impossible to keep up the hard work and finish the essay on time.

ABSTRACT

This essay focuses on two important technologies, Web services and agent based computing. It presents the state of the art of both technologies and then brings light on advantages that agent based computing has over Web services. Based on the research that has been done and the information collected, an agent based architecture for consuming Web services is proposed. It is worth mentioning that the work presented in this essay has two main objectives: to clearly explain the benefits and advantages of using agent based technologies and to propose a solution that supports these key benefits. The essay also tackles the issue of standardization and possible obstacles that these emerging technologies are facing and suggests for future research work that can be done. On personal level, working in this research area was an amazing experience for me. The benefits were multi-fold. I have learned how to collect, organize and filter the information. On the technical side I went through hundreds of relevant articles and the learning experience was unique. My writing skills are also improved a lot since all the time I had to think on how to present my thoughts in an easy to read and follow manner.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Lara Sauer, Gerry Drappel, and Larbi Esmahi for their continuous support and encouragement. Their interest, suggestions and advice meant a lot to me. Thank you all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Web services have become a significant technology in the evolution of the Web and distributed computing. Web services are able to improve corporate-software development by reducing the time and expenses involved in developing such software applications. Web services represent a set of related standards that allow any two applications to communicate and exchange data via the Internet.

Web services take advantage of object-oriented programming techniques and enable developers to build applications by reusing existing software components. This can significantly reduce the effort required to implement different types of systems. Web services also improve distributed computing capabilities by addressing the issue of limited interoperability. Web services operate using open standards, unlike DCOM and CORBA. This means that Web services can, theoretically speaking, enable any two software components residing in any two hardware platforms to communicate regardless of differences in programming languages or platforms.

New types of applications are being created every day by using standard Web services building blocks, and in this way greater economy of scale in automating business and consumer interactions with the Web and with each other.

Statement of the Purpose

The main focus of this paper is to bring some light on current state of the art on agent technology and how it can be used for implementing Web services as a way of communicating among different applications on disparate platforms. The agent technology fits well for Web services, since it supports dynamic scalability, distribution of services, reduction of traffic and independence regarding failures.

Intelligent and mobile agents can offer a good alternative to integrate some of the following properties to existing Web services:

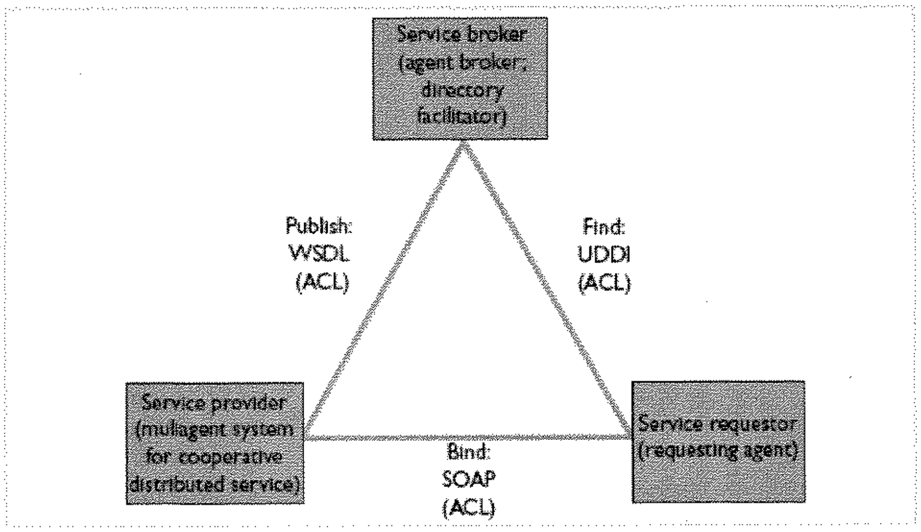
- ***Autonomy*** - Allows for making decisions on service access, the interface configuration, and service provisioning without human assistance. It also allows for automating the control and management tasks as well as automating the service deployment and provision.
- ***Personalization*** - Allows for the dynamic customization and configuration of services. The agents can learn and adapt to the preferences of their users and detect and update old versions of services.
- ***Mobility/Context awareness*** - Supports the dynamic topology of service provisioning. It enables Web-services to be provided instantly and customized directly at the locations where a service is needed.
- ***Cooperation/Sociability*** - Offers the potential to distribute service-related processing and also offers a mechanism for the nodes in different networks to cooperate in order to provide a service to the user. Allows for the asynchronous and cooperative processing of tasks.

Web services are currently based on the triad of functionalities (Huhns, 2002). As shown in Figure 1 (Huhns, 2002), these functionalities are:

- ***Publish*** - The Web Services Description Language (WSDL) describes the services in a machine-readable form, where the names of functions, their required parameters, and their results can be specified.
- ***Find*** - Universal Description, Discovery, and Integration (UDDI) gives clients, users and businesses a way to find needed services by specifying a registry or "yellow pages" of services.
- ***Bind*** – Finally, the simple object access protocol (SOAP) provides the common protocol systems need to communicate with each other so that they can request services, such as to schedule appointments, order parts, and deliver information.

The equivalent agent-based functionalities are shown in parentheses, and all interactions are via an agent-communication language (ACL), which makes the communication process simpler to handle.

Figure 1. Web services rely on the functionalities of publish, find, and bind.


Service broker (agent broker; directory facilitator)
Publish: WSDL (ACL)
Find: UDDI (ACL)
Service provider (mulagent system for cooperative distributed service)
Bind: SOAP (ACL)
Service requestor (requesting agent)

Limitations

In the world of Web services, the major industry software vendors have already agreed on the core standards. Microsoft, IBM, Sun Microsystems, BEA Systems, Oracle and other companies have agreed on implementing SOAP, WSDL, and UDDI. Additional technologies may or may not become part of the standard. Agent technologies are still new and it will take time and effort for the industry to adopt and use them with Web services.

Organization of the Thesis Essay

Aside from the Introduction chapter, the essay is spread out in five other chapters, which are briefly described in the following sections.

Overview of Web Services. This chapter covers the fundamentals of Web services technology and different standards used in industry today such as XML, WSDL, UDDI, and SOAP. It also describes the purpose of Web services and how they compare with existing similar technologies like distributed computing.

Agent Based Software Development. This chapter provides an overview of the state of the art in agent based software development. It also describes current standards in use as well as available agent oriented Integrated Development Environments (IDE).

Agent Based Web Services. This chapter gives a review of the current state of the art in implementing web services using agents. It presents more of a descriptive approach of what has been done in this field rather than presenting any conclusions or recommendations. The material in this chapter provides a sound foundation for the next chapter.

Discussion and Findings. This chapter presents the main benefits for using intelligent agents to implement Web services. It focuses on issues that can be solved only using agents or can be enhanced if implemented using agents. It also summarizes the data gathering process used and also shows how the various sources of information were analyzed. The last section presents original views in this field of research with a simple proposed architecture.

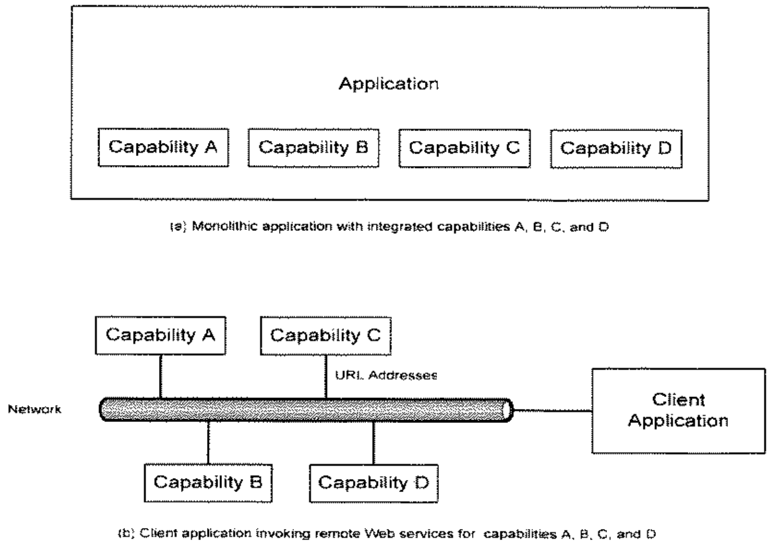
Conclusions and Recommendations. This chapter presents the conclusions after the research. It also provides recommendations related to the topic as well as future research areas in this field.

CHAPTER II

OVERVIEW OF WEB SERVICES

Web services represent a new architectural paradigm for applications. Web services implement different capabilities that are available to other applications (or other Web services) via industry standard network and application interfaces and protocols. An application can use the capabilities that a Web service offers by simply invoking it across a network without having to integrate it. This means that Web services act as reusable software building blocks which are also URL addressable. Figure 2 depicts the architectural differences between monolithic, integrated applications and Web services-based applications.

Figure 2. The architectural differences between (a) a monolithic application with integrated capabilities, and (b) a distributed application using Web services-based capabilities.


Application
Capability A
Capability B
Capability C
Capability D
(a) Monolithic application with integrated capabilities A, B, C, and D
Capability A
Capability C
URL Addresses
Network
Client Application
Capability B
Capability D
(b) Client application invoking remote Web services for capabilities A, B, C, and D

Although organizations are just beginning to implement and consume Web services, the basic standards and ideas have existed for several years. In 1999, Hewlett-Packard became the first software vendor to introduce the concept of Web services. HP's product, e-Speak, was a platform that enabled developers to build and implement "e-services", which were program units very similar to Web services. Because of the proprietary nature of e-Speak underlying technologies, this platform never gained widespread industry support.

Microsoft was the first company to use the term "Web services" in June 2000, when they introduced Web services as a key component of its .NET initiative. Today nearly every major software vendor is marketing Web services tools and applications.

Next are presented some key points that clarify the advantages associated with Web services.

- Web services operate using open, text-based standards, which enable components written in different languages and for different platforms to communicate.
- Web services promote a modular approach to programming, so multiple organizations can communicate with the same Web service.
- Web services are comparatively easy and inexpensive to implement, because they use existing infrastructure to exchange information. On top of that, most applications can be repackaged as Web services, so companies do not have to adopt entirely new software.

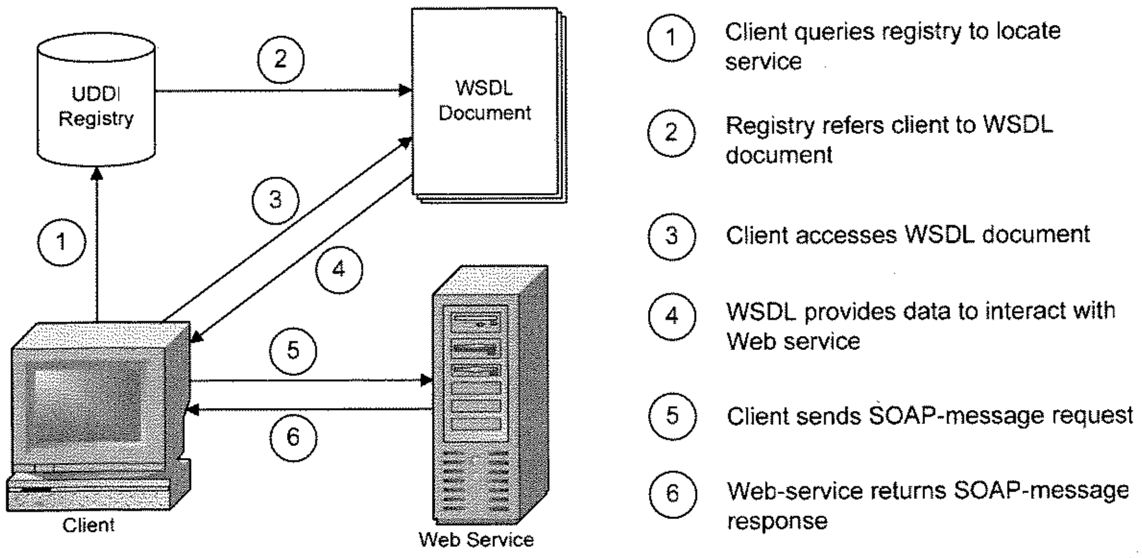
- Web services can significantly reduce the cost of enterprise application integration (EAI) and Business-To-Business (B2B) communications, and therefore offering companies tangible returns on their investments. (Borck, 2002).
- Web services can be implemented in an incremental manner, rather than all at once. This lowers the cost of adopting Web services and can reduce organizational disruption resulting from the process of switching to a new technology.

The most important advantage of Web services over previous distribute computing technologies is that they employ open standards. Because of the fact that Web services facilitate communications among disparate applications and platforms, standardization and interoperability are crucial. World Wide Web Consortium (W3C - an organization that defines Web technologies) and other standards bodies are committed to ensuring that Web services protocols and specifications remain open and interoperable across vendor implementations.

Part of what distinguishes Web services from similar computing models is the use of XML and XML-based standards – most commonly SOAP, WSDL and UDDI. These technologies enable communication among applications and hardware platforms. These technologies enable communication among applications in a manner that is independent of specific programming languages, operating systems and hardware platforms.

Figure 3 depicts the role of various standards in common Web services architectures (McGarr, 2002).

Figure 3. SOAP, UDDI and WSDL in a Web service interaction


UDDI Registry
WSDL Document
Client
Web Service
1
2
3
4
5
6
1 Client queries registry to locate service
2 Registry refers client to WSDL document
3 Client accesses WSDL document
4 WSDL provides data to interact with Web service
5 Client sends SOAP-message request
6 Web-service returns SOAP-message response

The following sections will briefly describe each of the technologies mentioned above starting with a short overview of XML.

XML (Extensible Markup Language)

Developed based on the Standard Generalized Mark-up Language (SGML), XML is a widely accepted standard for describing data and creating mark-up languages. Unlike many other technologies, which begin as proprietary solutions and later become standards, XML was defined by the W3C as an open, standard technology. In 1998, the XML version 1.0 specification was accepted as a W3C Recommendation, which means that the technology is stable for deployment in industry (World Wide Web Consortium, 2004).

XML is the foundation on which Web services are built. XML provides the description storage, and transmission format for data exchanged via Web services. XML is also used to create the Web services technologies that exchange data.

XML is similar in nature to the Hypertext Mark-up Language (HTML), having elements, attributes, and values. Well formed XML documents can be displayed in browsers, although this aspect of XML is not relevant to Web services. A major difference between HTML and XML is that HTML contains a finite set of elements and attributes, but XML allows any number of them to be defined.

XML elements and attributes independently define type and structure information for the data they carry, including the capability to model data and structure specific to a given software domain. The term software domain here is used to represent any programming language, a middleware system, a packaged application, or a database management system. XML-aware applications and tools parse, map, and transform generic XML data types into and out of software domain-specific types. Transforming a generic XML representation of data into an application, or a software domain-specific representation of data, is an essential aspect of Web services.

Two broad categories of XML usage in Web services are:

1. Data storage representation and format
2. Specification of the software that manipulates the data

For use in Web services, data can either be created in XML or converted to XML from one or more existing formats, such as ASCII or the Java type system. For example let us consider a fictive retail company which identified that the following basic data type information is required for a customer record in an ASCII file:

Customer ID	Integer
Customer Name	Character
Customer Address	Character
Customer Phone	Numeric
Postal Code	Character
E-Mail Address	Character
Credit Limit	Decimal
Credit Rating	Integer

After collecting this information from customer service department of the fictive retail company the analysts can format the data in XML as follows:

```
<Customer>
        <CustomerID>12345</CustomerID>
        <CustomerName>James Bond</CustomerName>
        <CustomerAddress>007 Secret Agent Drive</CustomerAddress>
        <CustomerPhone>777-777-7007</CustomePhone>
        <PostalCode>J0B-0S7</PostalCode>
        <EMailAddress>jbond007@secretplace.spy</EMailAddress>
        <CreditLimit>100000</CreditLimit>
```

```
    <CreditRating>10</CreditRating>
</Customer>
```

The Customer element is created containing all the data items. After representing the customer data as XML, the fictive retail company needs to create the XML schema to validate the customer information to ensure that it would have the correct structure and data types. XML schemas are a long topic and also outside of the scope of this essay, although the later chapters will refer to them.

SOAP (Simple Object Access Protocol)

SOAP is one of the common standards used to deliver Web services. Initially developed by representatives from DevelopMentor, Userland Software and Microsoft, SOAP was conceptualized in 1998 and published as SOAP 0.9 in 1999 (Benfield, 2001). After several versions released from above companies, the protocol was submitted to the W3C. The latest version of SOAP, SOAP 1.2, is currently being defined by W3C.

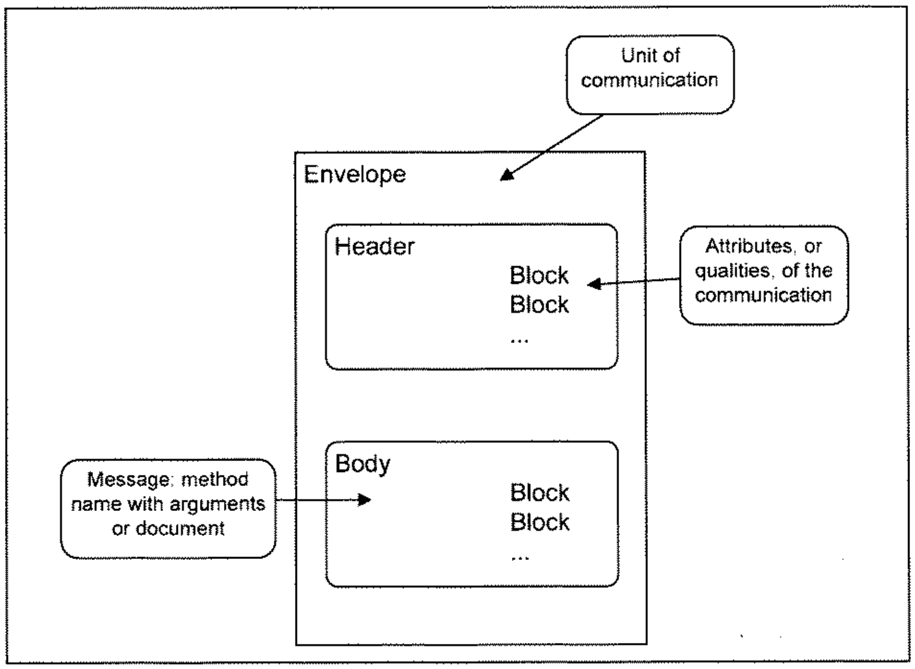
The purpose of SOAP is to enable data transfer between distributed systems. When an application communicates with a Web service, SOAP is the most commonly used standard through which data is exchanged. A SOAP message that is sent to a Web service invokes a method that is provided by that service, which means that the message actually requests that the service executes a particular task. The service uses information that is provided by the SOAP message to perform its action. If needed, the Web service can return the result back via another SOAP message.

SOAP is an XML-based communication protocol and it basically consists of a set of standardized XML Schemas. The Schemas define a format for transmitting XML messages over a network, including the types of data that the message can include and the way in which the message must be structured so that the server hosting the Web service can interpret it correctly (Howerton, 2002).

SOAP is layered over an Internet protocol, such as HTTP, and can be used to transfer data across the Web and other networks. The use of HTTP allows Web services to communicate across firewalls, because most firewalls are designed to accept HTTP service requests.

As shown in figure 4, SOAP consists of three main parts: an envelope, a header and a body (Newcomer, 2002) .

Figure 4. SOAP consists of three parts


Unit of communication
Envelope
Header
Block
Block
...
Attributes, or qualities, of the communication
Body
Block
Block
...
Message: method name with arguments or document

The envelope wraps the entire message and contains the header and body elements. The header is an optional element that provides information related to security, routing, etc. The body of the SOAP message contains the application-specific data that is being communicated. The data is marked up as XML and adheres to a specific format, which is already defined by the Schemas mentioned earlier. SOAP messages are received and interpreted by SOAP servers, which trigger Web services to perform their tasks.

SOAP is not the only protocol that can enable Web services. For example, XML-RPC is an older technology that provides similar functionality. It is important to mention that most major software vendors have already chosen to support SOAP over other similar technologies. A note worth mentioning is the advantages of SOAP (Cauldwell et. al., 2001):

- ***Simplicity*** - Most basic SOAP messages involve small amounts of code, and usually there is no need for special software to send and receive SOAP messages.
- ***Extensibility*** – SOAP provides mechanisms that allow developers to extend the standard in order to meet specific needs.
- ***Interoperability*** - Because SOAP uses XML to communicate over HTTP, it can theoretically be used to transfer data between any two systems that are connected to the Internet, regardless of the programming languages, operating systems and hardware platforms.

The following example shows a simple application of SOAP, which is a one way broadcast message to a list of communication mechanisms (PDA, Cell Phone, Email etc.). The header block contains the list of devices to which the message will be sent. The body block contains the actual notification message to be delivered.

```
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope>
  <env:Header>
        <n:broadcastService xmlns:n= “http://www.xyz.com/broadcastServices”>
                <n:list>PDA, Cell, Email, VoiceMail, IM</n:list>
        <n:broadcastService>
  </env:Header>
  <env:Body>
        <m:Function xmlns:m=“http://www.xyz.com/broadcastServices/send”>
                <m:message>Peter, you are late for the concall again!</m:message>
        </m:Function>
  </env:Body>
<env:Envelope>
```

In this example, the meeting reminder message will be broadcast to the listed devices by the “send” Web service located at the following address: http://www.xyz.com/broadcastServices. Separate namespaces (broadcastService and Function) are used to qualify the element and attributes for each part of the message. The envelope references version 1.2 envelope namespace, www.w3.org/2001/12/soap-envelope.

<u>WSDL (Web Services Description Language)</u>

Another important standard in enabling Web services is WSDL. An important feature of Web services is that they are self-describing. This means that every Web service is accompanied by information that enables developers to invoke that service. These descriptions typically are written in WSDL, which is an XML-based language through which a Web service can expose to other applications the methods that the service provides and how those methods can be accessed.

When SOAP and other Web services technologies were first developed, software vendors realized that applications calling services across a network would need information about a specific service before interacting with it. However, each vendor began building its own method of description, resulting in service descriptions that were incompatible with one another. The WSDL specification emerged when Microsoft and IBM decided to combine their description technologies into a universal standard. In March 2001, Microsoft, IBM and Ariba submitted WSDL 1.1 to the W3C. Currently the W3C is working on a version 2.0 of the language (World Wide Web Consortium, 2004). Although the technology is still under development, nearly all Web services products now provide support for WSDL 1. 1.

Most Web services published on the Internet are accompanied by an associated WSDL document, which lists the service's capabilities, states its location on the Web and provides instructions regarding its use. A WSDL document defines different types of messages a Web service can send and receive, as well as specifies the data that a calling application must provide for the Web service to perform its task. WSDL documents also provide specific technical information that informs applications about how to connect to and communicate with Web services over HTTP or another communications protocol.

It is important to realize that WSDL is a language meant to be read by applications, rather than by computer users. Although the structure of WSDL documents might appear complex, applications capable to understand WSDL can process the documents and extract the information they need. Furthermore, most Web services development tools generate WSDL documents automatically. This means that, if a programmer develops a Web service, the software used to build the service creates an appropriate WSDL document for that service automatically. Therefore, it is not necessary for developers to understand the syntax of WSDL fully when building and deploying Web services.

UDDI (Universial Description, Discovery and Integration)

The third major Web services standard, UDDI, enables developers and businesses to publish and locate Web services on a network. UDDI is originally designed by Microsoft, IBM and Ariba, and it initially started as a way for companies to share information about their businesses and business processes with potential partners and affiliates (Korzeniowski, 2002). UDDI specification allows companies to describe their own services and electronic processes, discover those of other companies and also integrate others services into their systems. Although UDDI is a relatively new standard (the first version was published in September 2000), it has acquired significant industry backing. UDDI version 2.0 was released in June 2001 while UDDI 3.0.1, the latest version, was released in October 2003.

UDDI defines an XML-based format in which companies can describe their electronic capabilities and business processes. The specification also provides a standardized method of registering and locating the descriptions on a network, such as the Internet. Part of the information that companies can supply is data regarding available Web services. Companies can store their information either in private UDDI registries, which are accessible only to approved business partners, or in public UDDI registries, which any interested party can use. The largest, most comprehensive public UDDI registry is the UDDI Business Registry (UBR), which was developed to facilitate the formation of new business relationships. Microsoft and IBM host implementations of the UBR that adhere to UDDI 1.0, and Microsoft, IBM, Hewlett-Packard and SAP host UBR implementations that adhere to UDDI 2.0. A UDDI registry's structure is conceptually similar to that of a phone book. Registries contain "white pages", where companies list contact information and textual description of themselves; "yellow pages," which provide classification information about companies and details on companies' electronic capabilities; and "green pages," which list technical data relating to services and business processes (Wilson, 2001). Information regarding businesses and services is highly categorized, enabling companies to search for desired partners or services. Information technology staff could use the technical information in the registries to link electronically to other businesses. In this manner, UDDI simplifies the process of creating B2B relationships and connecting electronic systems to exchange data and services.

Many companies have shown strong interest in UDDI registries. Most of the organizations that provide Web services support UDDI and have incorporated the standard into their products. For example, NTT Communications of Tokyo is in the process of building additional implementations of the UDDI Business Registry. However, businesses have been slow to enter information in the public registries. Industry experts believe that companies will begin using these services by building private registries to share services with partners. Large organizations can also create private registries to organize their own Web services and make the services available to other departments. Once the technology has matured and users are comfortable using it, public exchange of data and services will become more popular.

It is difficult for large organizations to change the way in which they communicate, form partnerships, locate clients and transact business. Some companies are hesitant to abandon older B2B communication mechanisms, whereas others are concerned about the security issues raised by exposing corporate data or applications on the Web. However, organizations are slowly realizing that technologies such as UDDI can improve business processes and provide competitive advantages.

Microsoft .NET Web Services Basics

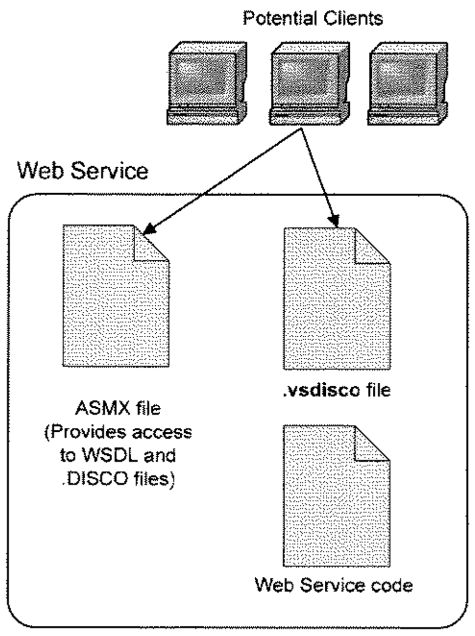
If you are developing a web application using .NET framework, a Web service is an application stored on one machine that can be accessed by another machine over a network. In its simplest form, a Web service is a class, or a logical grouping of methods that simplifies program organization. Methods are defined within a class to perform tasks and return information when their tasks are complete. .NET Web service classes contain certain methods (called Web service methods) that are specified as part of the Web service. These methods can be invoked remotely using either document-style or RPC-based messaging.

Creating Web services in Visual Studio .NET is a simple process. A developer first creates a project of type ***ASP .NET Web Service***. Among several files that are automatically generated by Visual Studio .NET, the following are specific to Web services applications (see Figure 5 below):

- ASMX file – provides documentation for the Web service
- DISCO file – potential clients use this file to discover the Web service
- Web service code file – contains all the code that provides the functionality of the Web service. The code can be written in any .NET compatible language (VB .NET, C++ .NET, C# .NET etc.).

Once the developer adds the necessary programming logic to the Web service code file and successfully compiles the application, then a client application can consume the Web service. However, clients must be able to find the Web service and learn about its capabilities.

Figure 5. .NET Web service components



Discovery of Web services (DISCO) is a Microsoft-specific technology used to locate Web services in a particular directory on a server. There are three types of discovery files: **.disco** files, **.vsdisco** files and **.map** files. The ASMX file (which is in the form of an ASP .NET Web page) can be viewed in a Web browser and contains descriptions of Web service methods and ways to test these methods.

Java Web Services Basics

The Java language and Java 2 Enterprise Edition (J2EE) platform provide features for building and deploying Web services. Two main benefits of the Java language and the J2EE platform are vendor independence and application portability. Applications built on the J2EE platform may be deployed on J2EE implementations from a large number of vendors. Using Java to develop Web services provides the benefit of vendor independence in addition to the inherent platform independence of Web services.

Java provides support for Web services through the Java Web Services Developer Pack (Java WSDP). Java WSDP contains libraries for generating XML and SOAP, processing XML, accessing service registries, and calling RPC-based Web services. Java WSDP provides Java developers with a one-stop API solution for the development of Java Web service applications.

Java WSDP brings together a set of Java APIs for XML-based Java applications by supporting key XML standards such as SAX, DOM, XSLT, SOAP, WSDL, UDDI, and ebXML. These APIs are bundled together with a set of runtime tools which allow Java WSDP to provide a build, deploy, and test environment for Web services applications and components. The pack includes the following toolset (Sun Microsystems, 2005):

- ***Java XML Pack*** - is an architectural solution toolkit that is intended to ease software development by providing a set of high level APIs and reference implementations that abstract the complexities behind XML processing.
- ***JavaServer Pages Standard Tag Libraries*** - is an initiative to standardize on a single set of reusable tag libraries that expose functionality to solve common problems faced in Web application development.

- ***Apache Tomcat container*** - is an open-source implementation of a Web container under the Apache Software Foundation. The container conforms to the latest specifications and provides runtime services for hosting and executing Servlets and JSPs.
- ***Java WSDP Registry Server*** - is an implementation of the UDDI version 3.0. The Java WSDP Registry Server serves the purpose of testing applications written using Java API for XML Registries (JAXR).
- ***ANT Build Tool*** - is a build tool similar to ***make*** and ***gnumake***. It has gained a lot of attention and acceptance from the community for building and deploying Java code. ANT uses XML for specifying the various tasks that must be executed in the build process. It provides many defined tasks that can be used by the developer while compiling, building, or deploying the application code.

Java WSDP is available at the Sun Java site. Full documentation, which explains the functionality of each Java XML API included in the pack, also is available for download. In addition, the Web services tutorial is also available for download. This tutorial takes the developer through each API with examples, and it provides instructions on how to set up the Java WSDP environment and how to deploy and test Web service applications.

The latest version of Java WSDP is version 1.5 and is available at http://java.sun.com/webservices/webservicespack.html.It includes the Java XML APIs and the runtime environment.

These development packs are released by Sun Microsystems on a quarterly basis, which ensures support for emerging XML standards and the most recent specifications.

CHAPTER III

AGENT BASED DEVELOPMENT FUNDAMENTALS

Since the 1980s, software agents and multi-agent systems have grown into what is now one of the most active areas of research and development activity in computing generally. There are many reasons for the current intensity of interest, but certainly one of the most important is that the concept of an agent as an autonomous system, capable of interacting with other agents in order to satisfy its design objectives, is a natural one for software designers. Just as we can understand many systems as being composed of essentially passive objects, which have state, and upon which we can perform operations, so we can understand many others as being made up of interacting, semi-autonomous agents.

What are Agent-Based Systems?

Before proceeding any further, it is important to gain an understanding of exactly what is meant by an agent-based system. An agent-based system is a system in which the key abstraction used is that of an agent. Agent-based systems may contain a single agent, (as in the case of user interface agents or software secretaries), but arguably the greatest potential lies in the application of multi-agent systems. Agent-based system is a system that enjoys the following properties:

- **Autonomy -** agents encapsulate some state (that is not accessible to other agents), and make decisions about what to do based on this state, without the direct intervention of humans or other agents.
- **Reactivity -** agents are situated in an environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps many of these combined), are able to perceive this environment, and are able to respond in a timely fashion to changes that occur in it.
- **Pro-activeness** - agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.
- **Sociability -** agents interact with other agents (and possibly humans) using an agent communication language, and typically have the ability to engage in social activities (such as cooperative problem solving or negotiation) in order to achieve their goals.

These properties are more demanding than they might appear at first. To see why, let us consider them in turn. Let us consider pro-activeness which can be described as goal directed behaviour. It is not hard to build a system that exhibits goal directed behaviour because most of the systems today are based on procedures written in Visual Basic, functions written in C++, or methods written in Java. When we write such a procedure, we describe it in terms of the assumptions on which it relies (it's pre-condition) and the effect it has if the assumptions are valid (it's post-condition). The effects of the procedure are its goal, for example what the author of the software intends the procedure to achieve. If the pre-condition is true when the procedure is invoked, then we expect that the procedure will execute correctly, after that it will terminate, and that upon termination, the post-condition will be true, i.e., the goal will be achieved. This is goal directed behaviour and the procedure is simply a plan or recipe for achieving the goal. The presented programming model works fine for most environments. For example, it works well when we consider functional systems, those which take some input ***x***, and produce as an output some function ***f(x)*** of this input. Compilers are a classic example of functional systems (Wooldridge & Ciancarini, 2000).

But for non-functional systems, this simple model of goal directed programming is not acceptable, as it makes an important limiting assumption. It assumes that the environment does not change (static environment) while the procedure is executing. If the environment does change (dynamic environment), and in particular, if the assumptions (pre-condition) underlying the procedure become false while the procedure is executing, then the behaviour of the procedure may not be defined which often leads to its premature termination (procedure crash). Similarly, it is assumed that the goal, which also is the reason for executing the procedure, remains valid at least until the procedure terminates. If the goal does not remain valid, then there is simply no reason to continue executing the procedure. In many environments, neither of these assumptions is valid. In particular, in domains that are too complex for a single agent to observe completely, also known as multi-agent systems (a system that is populated with more than one agent that can change the environment), or where there is uncertainty in the environment, these assumptions do not stand. In such environments, blindly executing a procedure without any consideration to whether the assumptions related to the procedure are valid is a poor strategy. In such dynamic environments, an agent must be reactive, in just the way that we described above. This means that it must be responsive to events that occur in its environment. These events affect either the agent's goals or the assumptions related to the procedures that the agent is executing in order to achieve its goals.

In general, building purely goal directed systems is not hard. Similarly, building purely reactive systems, ones that continually respond to their environment, is also not difficult. It is possible to implement them as lookup tables that simply match environmental stimuli to various action responses. However, what turns out to be very hard is building a system that achieves an intelligent balance between goal-directed and reactive behaviour. It is desirable to have agents that will attempt to achieve their goals systematically, perhaps by making use of complex procedure-like recipes for action. On the other hand we do not want our agents to continue blindly executing these procedures in an attempt to achieve a goal when it is clear that the procedure will not work, or when the goal is for some reason no longer valid. In such circumstances, we want agents to be able to react to the new situation, in time for the reaction to be of some use. However, we do not want our agent to be reacting in continuous mode, and never focusing on a goal long enough to actually achieve it. This problem of effectively integrating goal-directed and reactive behaviour is one of the key problems facing the agent designer. Many proposals have been made related to building such agents, but the problem still remains open.

Next social ability of agents is discussed, one of the component of flexible autonomous action. Social ability is trivial if it is seen in a narrow, one-sided view: every day, millions of computers across the world routinely exchange information with both humans and other computers. This ability to exchange bits of information does not represent real social ability. The type of social ability which involves the ability to dynamically negotiate and coordinate is much more complex, and much less understood, than simply the ability to exchange bit streams.

An obvious question to ask is why agents and multi-agent systems are seen as an important new direction in software engineering. There are several reasons:

- **Natural metaphor** - Just as the many domains can be conceived of consisting of a number of interacting but essentially passive objects, so many others can be conceived as interacting, active, purposeful agents. For example, a scenario currently driving a lot of research activity in the agent field is that of software agents capable of buying and selling goods via the Internet on behalf of clients. It is natural to view the software participants in such transactions as semi-autonomous agents.
- **Distribution of data or control** - For many software systems, it is not possible to identify a single point of control. Instead, overall control of the systems is distributed across a number computing nodes, which are frequently distributed geographically. In order to make such systems work effectively, these nodes must be capable of autonomously interacting with each other. These nodes must be implemented as agents.

- **Legacy systems -** A natural way of incorporating legacy systems into modern distributed information systems is to provide another abstraction layer on top of them, which can be done by wrapping them with an agent layer that will enable them to interact with other agents.
- **Open systems -** Many systems are open in the sense that it is impossible to know at design time exactly what components the system will be comprised of, and how these components will be used to interact with each other. To operate effectively in such systems, the ability to engage in flexible autonomous decision making is critical.

Relationship between Agents and Objects

Programmers familiar with object-oriented approach often fail to see anything new in the idea of agents. When one stops to consider the relative properties of agents and objects, this is perhaps not surprising. Objects are defined as computational entities that encapsulate some state, are able to perform actions, or methods on this state, and communicate by message passing. There are clearly close links between agents and objects.

While there are obvious similarities, there are also significant differences between agents and objects. The first is in the degree to which agents and objects are autonomous. One of the defining characteristics of object-oriented programming is the principle of encapsulation. In programming languages like Java, it is possible to declare instance variables (and methods) to be private, which means they are only accessible from within the object (we can of course also declare them public, meaning that they can be accessed from anywhere, and indeed we must do this for methods so that they can be used by other objects). In this way, an object can be thought of as exhibiting autonomy over its state: it has control over it. But an object does not exhibit control over its behaviour. That is, if an object has a public method ***m***, then other objects can invoke ***m*** whenever they wish, once an object has made a method public, then it subsequently has no control over whether or not that method is executed.

Of course, an object must make methods available to other objects, or else we would not be able to build a system based on them. Declaring object's methods as public does not create any issues, because if we build a system, then we design the objects that go in it, and they can be assumed to share a "common goal". But in many types of multi-agent systems, (in particular, those that contain agents built by different organizations or individuals), no such common goal can be assumed. It cannot be for granted that an agent ***x*** will execute an action (method) ***a*** just because another agent ***y*** wants it to, because ***a*** may not be in the best interests of ***x***. Therefore, we do not think of agents as invoking each others methods, but rather as entities that request actions to be performed. If ***y*** requests ***x*** to perform ***a***, then ***x*** may perform the action or it may not. The level of control with respect to the decision about whether to execute an action is therefore different in agent and object systems. In the object-oriented case, the decision lies with the object that invokes the method. In agent-based systems, the decision lies with the agent that receives the request. This distinction between objects and agents has been nicely summarized in the following slogan: “Objects do it for free; agents do it because they want to” (Wooldridge & Ciancarini, 2000).

The second important distinction between object-based and agent-based systems is with respect to the notion of flexible (reactive, pro-active, social) autonomous behaviour. The standard object model does not provide anything related to building systems that integrate this type of behaviour. One could point out that it is possible to build object-oriented applications that do integrate this type of behaviour. But this argument is off the point, because the standard object-oriented programming model has nothing to do with this type of behaviour.

The third important distinction between the standard object-oriented model and the general view of agent systems is that each agent is considered to have its own thread of control. Agents are assumed to be continuously active, and typically are engaged in an infinite loop of observing their environment, updating their internal state, and selecting and executing an action to perform. In contrast, objects are assumed to be inactive for most of the time, becoming active only when another object requires their services by using method invocation.

A lot of work has been recently devoted to concurrency in object-oriented programming. For example, the Java language provides built-in constructs for multithreaded programming. There are also many programming languages available (most of them as prototypes only) that were specifically designed to allow concurrent object-based programming. All these languages do not capture the idea of agents being autonomous entities. Perhaps the closest that the object-oriented community comes is using the concept of active objects.

An active object is one that encompasses its own thread of control. Active objects are generally autonomous, meaning that they can exhibit some behaviour without being operated upon by another object. Passive objects, on the other hand, can only undergo a state change when explicitly acted upon (Booch, 1994).

Therefore, active objects are essentially agents that do not necessarily have the ability to exhibit flexible autonomous behaviour.

To summarize this comparison, objects and agents have at least three distinctions:

- Agents embody stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from another agent.
- Agents are capable of flexible (reactive, pro-active, social) behaviour, and the standard object model has nothing to say about such types of behaviour.
- A multi-agent system is inherently multi-threaded, which means each agent is assumed to have at least one thread of control.

Agent-Oriented Methodologies

Agents have been increasingly recognized as the next candidate for software engineering approach. Methodologies are the means provided by software engineering to facilitate the process of developing software and, as a result, to increase the quality of software products. By definition, a software engineering methodology is a structured set of concepts, guidelines or activities to assist people in undertaking software development.

It is also important for a methodology to provide notations which allow the developers to model the target system and its environment. In addition to the methodology, there are also tools that support the use of such methodologies. For instance, diagram editors help the developers drawing symbols, models described in the methodology. The Rational Unified Process (RUP) is a good example of a software engineering methodology. It uses the notation described in the Unified Modelling Language (UML) and its typical tool support is Rational Rose.

Despite their current dominance, RUP, UML and other object-oriented methodologies are regarded as unsuitable to the analysis and design of agent-based systems. The main reason is the inherent differences between the two entities, agents and objects, as discussed in previous section. As a result, object-oriented methodologies generally do not provide techniques and model to encapsulate intelligent behaviour of agents. Therefore, there is a need for software engineering methodologies which are specially tailored to the development of agent-based systems.

In answering this demand, there have been an increasing number of agent-oriented methodologies proposed in recent years (see Table 1 in Appendix). A common property of these methodologies is that they are developed based on the approach of extending existing methodologies to include the relevant aspects of agents. They are broadly categorized into two groups: extensions of Object-Oriented methodologies and extensions of Knowledge Engineering frameworks.

Extensions of Object-Oriented (OO) methodologies extend existing OO methodologies or adapt them to the aim of Agent Oriented Software Engineering. There are several reasons for following this approach. First of all, agent-oriented methodologies which extend object-oriented design can benefit from the similarities between agents and objects. Secondly, they can capitalize on the popularity and maturity of OO methodologies. There is a high possibility that they will be accepted easier compared to other non OO methodologies. Finally, several techniques such as use cases and class responsibilities card (CRC) used for object identification can be used for agents with the similar purpose.

Extensions of Knowledge Engineering (KE) techniques cover aspects of agents that are not addressed in OO methodologies. For instance, OO methodologies do not define techniques for modeling the mental state of agents. In addition, the social relationship between agents can hardly be captured using OO methodologies. These are some of the arguments for adopting KE methodologies for AOSE. They are suitable for modelling agent knowledge due to the fact that the process of capturing knowledge is addressed by many KE methodologies. Additionally, existing techniques and models in KE such as ontology libraries, and problem solving method libraries can be reused in agent-oriented methodologies.

As shown in Table 1, there are a large number of agent-oriented methodologies available.

Agent-Oriented Development Tools and Platforms

A number of tools and platforms are available that support activities or phases of the process of agent-oriented software development. Most of them are built on top of and integrated with Java. While almost all available tools and platforms have their focus on implementation support, some of them do also support analysis, design, and test/debugging activities. It is beyond the scope of this essay to describe and compare the available tools and platforms in detail. However, in the following paragraphs some of the most prominent tools and platforms are listed. Examples of often cited academic and research prototypes are the following (Weiss, 2002):

- ZEUS is a toolkit that has been developed at the British Telecom Intelligent System Research Lab.
- JADE ("Java Agent DEvelopment Framework") has been developed at the University of Parma, Italy.
- LEAP ("Lightweight Extensible Agent Platform") is intended to be executable on small devises such as PDAs or phones. LEAP is being developed within European's Fifth Framework program by several industrial and academic contract partners (MOTOROLA, ADAC, BROADCOM, BT, Siemens, and the University of Parma).
- agenTool is a Java-based graphical development environment that supports the MaSE method. agenTool was originally developed at the Artificial Intelligence Lab of the Air Force Institute of Technology, Ohio.

- RETSINA is a complex environment for networked intelligent agents that include different (multi) agent architectures, location and discovery services, middle agents, and configuration management support. RETSINA has been developed at Carnegie Mellon University.
- JATLite ("Java Agent Template, Lite"), which has been developed at the Stanford Center for Design, is a package of Java programs that allows creating software agents that communicate over the Internet.
- FIPA-OS is a component-based toolkit for the development of FIPA compliant agents. Two types of FIPA-OS are available, namely, "standard" for execution on standard computers and "micro" for execution on PDAs.
- MADKIT is a platform which is being developed at LIRMM (France). MADKIT is based on the Aalaadin model.

Other examples are SIM AGENT, JAFMAS ("Java-based Agent Framework for Multi-Agent Systems"), ABS ("Agent Building Shell") which employs the language COOL, OAA ("Open Agent Architecture"), and Agentis which is a modeling framework for BDI agents.

Below are listed some examples of commercial products for developmental support:

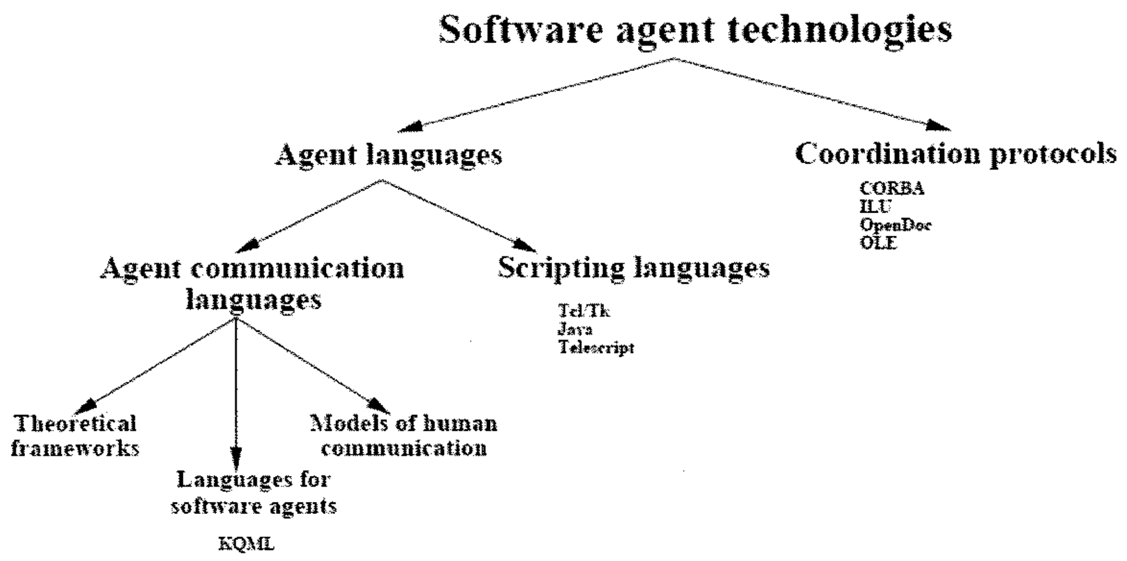
- AgentBuilder is a tool offered by Reticular Systems Inc., USA. AgentBuilder is available in two versions: AgentBuilder Lite (entry-level) and AgentBuilder Pro.
- JACK is a commercial agent framework by Agent Oriented Software Pty. Ltd., Melbourne, Australia. JACK is oriented towards BDI agents.

- Intelligent Agent Factory by Bits & Pixels, Texas, USA.
- Grasshopper is an advanced development platform for mobile agents launched by IKV++, Germany.

Agent Communication Language Fundamentals

A wide variety of systems, languages, frameworks and standards are associated with software agents. This is due in part to the vagueness of the term "software agent". This section represents an attempt to split apart these approaches, and show how Knowledge Query and Manipulation Language (KQML) relates to each of them. Figure 6 shows categorization of technologies important to software agents (Mayfield, 1996). We divide agent technologies into two broad categories: agent languages and coordination protocols.

Figure 6. A categorization of agent related technologies.



The agent languages category comprises all languages that can be used to implement software agents. Virtually any programming language can be used for software agent development. One class of languages that has gained much attention lately is the so called scripting languages class, especially those designed for mobile applications. Languages like Tcl/Tk, Java and Telescript offer the advantage of a level of abstraction that seems particularly attractive for the development of software agents. They are placed under the agent languages folder, because they can be used to program software agents. They differ from agent communication languages though because they are designed primarily to control processes on a single platform. To the extent that these languages contain communication primitives tailored to agent development, they are largely concerned with the transportation of a single agent from one machine to another (Mayfield, 1996).

In contrast, agent communication languages are designed specifically to describe and facilitate communication among two or more agents. Three broad sub-categories may be identified under the label of agent communication languages:

- Models of human communication
- Theoretical frameworks
- Communication languages for software agents

Human communication is traditionally modeled in terms of speech act theory. Considerable work has been done to capture the assumptions and conventions of interaction between human agents and subsequently translate them into workable paradigms for the development of their artificial counterparts. Often, such work leads to theoretical frameworks for artificial agents with human-like capabilities. Such frameworks attempt to account for all aspects of the internal state of an artificial autonomous agent, with a particular attention to how this state changes as the agent interacts (and/or communicates) with the world or with other agents. Sometimes, as in the case of Agent Oriented Programming, those frameworks may evolve into implemented software systems. In contrast, agent communication languages (ACL) are concerned strictly with the communication between such computational entities. An ACL (the sub-category that includes KQML) is more than a protocol for exchanging data, because information about what is exchanged by the agents is also communicated. An ACL may be thought as a communication protocol (or a collection of protocols) that supports many message types.

The other main class of software agent technologies is that of standards and coordination protocols. CORBA, ILU, OpenDoc, OLE, etc., are efforts that are often proclaimed as solutions to the agent communication problem. Driving such work is the difficulty of running applications in dynamic, distributed environments. The primary concern of these technologies is to ensure that applications can exchange data structures and methods across disparate platforms. Although the results of such efforts will be useful in the development of software agents, they do not provide complete answers to the problems of agent communication. After all, software agents are more than collections of data structures and methods on them. Thus, these standards and protocols are best viewed as a foundation on which agent languages might be built.

Limitations

AOSE is at an early stage of evolution. While there are many good arguments to support the view that agents represent an important direction for software engineering, there is a deficiency of actual experience to support these arguments. Preliminary methodologies and software tools to support the deployment of agent systems are beginning to appear, but slowly. This section, points out some of the key obstacles that must be overcome in order for AOSE to become "mainstream":

- *Sorting out the relationship of agents to other software paradigms - objects in particular.*

  It is not yet clear how the development of agent systems will coexist with other software paradigms, such as object-oriented development.

- *Agent-oriented methodologies.*

  Although a number of preliminary agent-oriented analysis and design methodologies have been proposed, there is comparatively little consensus between them. In most cases, there is not even agreement on the kinds of concepts the methodology should support. The presence of UML as the predominant modeling language for object-oriented systems and the kinds of concepts and notations supported by UML are not necessarily those best-suited to the development of agent systems.

- *Engineering for open systems.*

In open such systems, it is essential to be capable of reacting to unforeseen events, exploiting opportunities where these arise, and dynamically reaching agreements with system components whose presence could not be predicted at design time. However, it is difficult to know how to specify such systems, and even more so how to implement them. Therefore, a better understanding of how to engineer open systems is needed.

- *Engineering for scalability.*

  Finally, a better understanding is needed of how to safely and predictably engineer systems comprised of massive numbers of agents that dynamically interact with each other in order to achieve their goals. Such systems seem prone to problems such as unstable/chaotic behaviours, feedback, and so on, and may easily fall prey to malicious behaviour such as viruses.

CHAPTER IV

AGENT BASED WEB SERVICES

While a Web service does not have to possess all characteristics of an agent, the Web services approach to building complex software systems bears many similarities to the engineering process of a collection of software agents. In particular, large systems are assembled from distributed heterogeneous software components providing specialized services and communicating using protocols based on previous mutual agreement. Similarly to certain multi-agent engineering paradigms, the design process of such systems focuses on the declarative characterization of the capabilities of agents and on a message-based paradigm of interoperation.

The area of Web services offers real interest to the Multi Agent community, mainly because of similarities in system architectures, powerful tools, and the focus on issues such as security and reliability. Similarly, techniques developed in the Multi Agent research community promise to have a strong impact on this fast growing technology.

One of the best sources of information on the current state of research in this field is the yearly workshop on Web services and Agent-Based Engineering (WSABE). This year's workshop was held in New York City on July the 19th, and there were eight accepted papers (four short papers and four long ones). Among other papers which focused more on applied side is work submitted by Ishikawa, Yoshioka and Tahara which brings light on the process of synthesis of Web services and mobile agents (Ishikawa et. al., 2004). This approach introduces Web services and mobile agents and presents original approach to so called Mobile Web services as synthesis of Web services and mobile agents.

To realize the potential of agents to manage interactions with Web services a number of research efforts are under way to bring semantics to Web service descriptions that will sit at layers above what is being offered commercially. A number of approaches have been offered to provide Web services with agent-like behaviour through the use of agent wrappers. Some approaches use wrappers so that web sources can be queried in a similar manner to databases (Buhler & Vidal, 2003).

The following sections will discuss the application of agents in three main areas of interest:

- Grid Computing
- Semantic Web
- Automatic composition of services

Agents and Grid Computing

The Grid and agent communities both develop concepts and mechanisms for open distributed systems, although they do that from different perspectives. The Grid community has historically focused on infrastructure, tools, and applications for reliable and secure resource sharing within dynamic and geographically distributed virtual organizations. In contrast, the agent community has focused on autonomous problem solvers that can act flexibly in uncertain and dynamic environments. Yet as the scale and ambition of both Grid and agent deployments increase, we see a convergence of interests, with agent systems requiring robust infrastructure and Grid systems requiring autonomous, flexible behaviours.

Grids provide an infrastructure for federated resource sharing across trust domains. Much like the Internet on which they build, current Grids define protocols and middleware that can mediate access provided by this layer to discover, aggregate, and harness resources. These applications span a wide spectrum. Moreover, the standardization of the protocols and interfaces used to construct systems is an important part of the overall research and development program.

Grid technologies have evolved through at least three distinct generations: early ad hoc solutions, de facto standards based on the Globus Toolkit (GT), and the current emergence of more formal Web services based standards within the context of the Open Grid Services Architecture (OGSA) (Foster et. al. 2002). The Grid community has participated in, and in some cases led, the development of Web services specifications that address other Grid requirements. The Web Services Resource Framework (WSRF) defines uniform mechanisms for defining, inspecting, and managing remote state, a crucial concern in many settings. WSRF mechanisms underlie work on service management (WSDM, in OASIS) and negotiation (Web Services Agreement, in GGF), efforts that are crucial to the Grid vision of large-scale, reliable, and interoperable Grid applications and services. Other relevant efforts are aimed at standardizing interfaces to data, computers, and other classes of resources.

A core unifying concept that underlies Grids and agent systems is that of a service: an entity that provides a capability to a client via a well-defined message exchange (Booth et. al., 2003). Within third generation Grids, service interactions are structured via Web service mechanisms, and thus all entities are services. However, while every agent can be considered a service (in that it interacts with other agents and its environment via message exchanges), we might reasonably state that not every Grid service is necessarily an agent (in that it may not participate in message exchanges that exhibit flexible autonomous actions).

The Grid state model defines how state is represented and accessed, but does not speak to the structure or semantics of the state that is exposed. Typical practice is to define state in terms of fixed schema or attributes. In contrast, agent systems address semantics but do not provide a consistent state model. An integrated approach can allow for the publication of richer semantic information within the Grid state model, thus enhancing the ability of applications to discover, configure, and manage services in an interoperable manner (Goble, De Roure, Shadbolt & Fernandes, 2004).

While Grid technologies provide the means for describing and grouping services, these higher level matchmaking and discovery capabilities are not currently part of Grid infrastructure. Fortunately, this is an area where much work has been done in the space of agents, and incorporation of this technology would help a lot to improve this situation. This integration may have an impact on how state is represented and how services are organized.

As with other aspects of agents and Grids, it is expected to see the adaptation of agent algorithms and technologies as they incorporate policy specification and enforcement into their basic operations and Grid algorithms making use of some of the richness of the various agent trust and reputation models that have been developed (Ramchurn, Huynh & Jennings, 2004). It is also expected that the types of policy statements made, as well as how are they disseminated and applied, will evolve as agent-based techniques become more completely integrated into Grids. For example, reputation-based authentication mechanisms, which lend themselves to agent-based implementations, show great promise in the Grid environment.

Semantic Web

One general characterization of the technologies used for service discovery in the Web services world can be made by studying the difference between approaches which could be considered semantically poor and those which are semantically rich. In the former case, services are often referred to by opaque names or function signatures which give little or no indication of the nature of the services being managed. In the latter, however, service descriptions are more complex expressions which are based on terms from agreed vocabularies, and which attempt to describe the meaning of the service, rather than simply assigning a name to it. A key component in the semantically-rich approach is the ontology, the formal, agreed vocabulary whose terms are used in the construction of service descriptions. An ontology is a conceptualization of an application domain in a human understandable and machine readable form, and typically comprises the classes of entities, relations between entities and the axioms which apply to the entities which exist in that domain. Ontologies are currently a fast growing research topic, with interest from several communities, including agent-based computing, Semantic Web and knowledge management communities, because they offer a more formal basis for characterizing the knowledge assets held by software agents, Semantic Web services or organizations (Guarino & Giaretta, 1995).

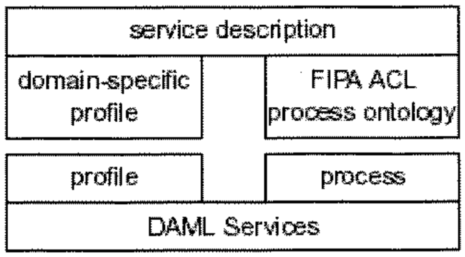
Although such ontology defines the agreed meaning for the application domain-specific terms used in the content of messages, it does not define the meaning of the message types themselves, or their effects upon the recipient. The current approach in the Semantic Web to Web services, such as that taken by DAML Services, does not provide a common basis for defining the pragmatics of different message types, as it might be expected from a speech act-like treatment of messages (Searle, 1969). Such a basis would provide a way to ease the introduction of new types of messages, since there would be a common understanding of what was meant by, for example, a directive message (which instructs a system component to perform an action) or an assertive message (which informs a system component of some fact) which was independent of any domain specific meaning. The technique of factoring out the common attributes of message types and ascribing them to different classes of speech acts is commonly used in the design of agent communication languages (ACL) for multiagent systems (FIPA, Oct. 2002), where there is a clear separation made between the domain-specific and domain-independent aspects of communication. A similar approach can be applied to Web services, in which an ACL component is integrated into the semantically rich service descriptions.

In the conventional Web services approach exemplified by WSDL or even by DAML Services, the communicative intent of a message (for example, whether it is a request or an assertion) is not separated from the application domain. This is at odds with the convention from the Multi-Agent Systems world, where there is a clear separation between the intent of a message, which is expressed using an agent communication language, and the application domain of the message, which is expressed in the content of the message by means of domain-specific ontologies.

This separation between intent and domain is beneficial because it reduces the brittleness of a system. If the characterization of the application domain (the ontology) changes, then only that component which deals with the domain-specific information needs to change, while the agent communication language component remains unchanged.

The division of service descriptions into a profile and a process component, as in DAML Services, provides a means to compartmentalize Web services in a manner similar to that found in agent systems. It therefore makes sense to describe the pragmatics of message types in the process component, giving an abstract ontology of message types that corresponds to the agent communication language, while the more application-specific details of the abilities of a particular agent (expressed as constraints on the content of messages) are expressed in the profile component, as shown in Figure 7.

Figure 7. Service Description with ACL Process Ontology

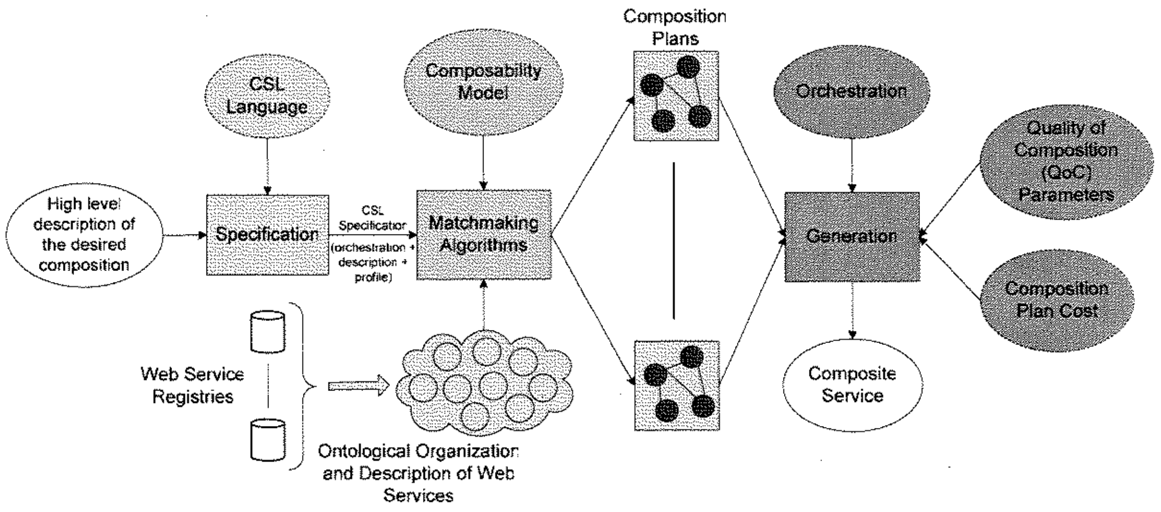


Automatic composition of services

The automatic composition of Web services is a recent trend that focuses on simplifying the process of composition of Web services and allowing services to automatically acquire related services so they can perform the task together (Berners-Lee, 2001). This section presents a short overview of Semantic Web centered approach for the automatic composition of Web services (Medjahed, Bouguettaya & Elmagarmid, 2003). One of the proposed approaches builds on the semantic description framework and composability model (Medjahed, 2004). It consists of three conceptually separate phases: specification, matchmaking, and generation (see Figure 8).

The specification phase (phase 1) enables high-level descriptions of composition requests. For that purpose, a Composition Specification Language (CSL) is defined. CSL extends UML activity diagrams with an ontological description of the composition request (Object Management Group, 1999). Using CSL, composers specify the ***what*** part of the desired composition but will not concern themselves with the ***how*** part.

Figure 8. Overview of one the Proposed Approaches for Automatic Service Composition


CSL Language
Composability Model
Composition Plans
Orchestration
Quality of Composition (QoC) Parameters
High level description of the desired composition
Specification
CSL Specification (orchestration + description + profile)
Matchmaking Algorithms
Generation
Composition Plan Cost
Web Service Registries
Composite Service
Ontological Organization and Description of Web Services

Also, composers are not required to be aware of the full technical details such as the list of participant services and the way they are executed and plugged together. The matchmaking phase (phase 2) uses the composability model to generate composition plans that conforms to composer's specifications. One or several composition plans are generated for each composer's sub-request. By composition plan, we refer to the list of imported operations to be outsourced, their orchestration order, and the way they interact with each other (plugging operations, mapping messages, etc.) in order to implement the corresponding sub-request. In the generation phase (phase 3), detailed descriptions of the composite service are generated. Quality of Composition (QoC) model is specified to assess the quality of the generated descriptions.

The following list presents research prototypes that support Web service composition focusing on a representative set of such prototypes.

- CMI (Collaboration Management Infrastructure) - CMI provides an architecture for inter-enterprise workflows. The main components of CMI engine include the CORE, coordination and the awareness engines (Schuster, Baker, Cichocki, Georgakopoulos & Rusinkiewicz, 2000).
- eFlow - eFlow is a platform that supports the specification, enactment, and management of composite services (Casati, Ilnicki, Jin, Krishnamoorthy & Shan, 2000).
- WISE (Workflow based Internet SErvices) - WISE aims at providing an infrastructure for the support of cross-organizational business processes in virtual enterprises (Lazcano, Schuldt, Alonso & Schek, 2001).
- CrossFlow - The main contribution of CrossFlow is in using the concept of contracts as a basic tool for cooperation (Ludwig & Hoffner, February 1999).
- Mentor-Lite - Mentor-Lite addresses the problem of distributing the execution of workflows (Weissenfels, Gillmann, Roth, Shegalov & Wonner, 2000).
- XL (XML Language) - XL defines an XML language for the specification of Web services. An XL service specification contains local declarations, declarative clauses, and operation specifications (Florescu, Grunhagen & Kossmann, 2002).
- SELF-SERV (compoSing wEb accessibLe inFormation and buSiness sERvices) - SELF-SERV proposes a process-based language for composing Web services based on state charts (Shen, Benatallah, Dumas & Mak, 2002).

In Table 2 (see Appendix), the aforementioned prototypes are compared using interaction layers (Medjahed, 2004). For example, eFlow uses RMI at the communication layer. At the content layer, eFlow provides adapters to support different interaction protocols such as OBI and RosettaNet. Interoperability at the business layer is enabled through a process description model based on state machines.

The same prototypes are compared in Table 3 (see Appendix) using key interaction dimensions such as Coupling, Autonomy, Heterogeneity, etc. (Medjahed, 2004).

For example, WISE allows only tight coupling among B2B participants. In terms of autonomy, partners must advertise services in encapsulated objects. Heterogeneous interaction protocols are supported through object-based middleware. External manageability and adaptability are possible via process monitoring/analysis module and execution guarantee. Security however is not addressed. Scalability is accommodated using distributed architecture.

CHAPTER V

DISCUSSION AND FINDINGS

The following sections of this chapter will focus on the benefits of using intelligent agents to implement Web services. They cover issues that can be solved only using agents or can be enhanced if implemented using agents. A short overview of the data gathering process used and how the various sources of information were analyzed follows. The last section focuses on my personal findings and also presents a proposed architecture for agent based Web services.

Agents to the rescue

The issues related to Web services where agents can provide a solution or better performance are:

- Cooperation
- Autonomy
- Context awareness
- Personalized services/Dynamic adaptation

Cooperation. Web services as we know them today cannot cooperate among each other without a manual (human) intervention or request. Cooperation among Web services would not be needed if all the services offered could do the job on their own without any help from other services. In reality this is rarely the case. Let us examine a simple situation where a Web service books an airline ticket. Without cooperation the Web service will only do what it is programmed to do which in this case is booking a ticket. If we implement this service by using agents and their cooperative nature we can provide a much more robust service to clients. In this example an agent based service could automatically request information from other agent based services about different flights available, prices, nearby hotels and rates, car rental availability and cost, attractions, currency exchange, weather forecast for the time the client will be away and many other services. This smart service can act as a multi-purpose broker for clients.

In order for Web services to cooperate among each other they need to use a language of some sort. That is where agents come in place. Agent Communication Language (ACL) provides agents with a means of exchanging information and knowledge. Knowledge Query and Manipulation Language (KQML) and the Foundation for Intelligent and Physical Agents (FIPA) ACL are two most widely used ACLs. A relatively new approach to agent communication is DARPA Agent Mark-Up Language (DAML) which can be used to encode ACL in DAML language (Zou, Finin, Peng, Joshi and Cost, 2003). Web services gain a lot more power and robustness with this added capability of cooperation among each other.

Autonomy. A Web service, as currently defined and used, is not autonomous. Autonomy is a characteristic of agents, and it is also a characteristic of many envisioned Internet based applications. One of the major drawbacks of the Web services infrastructure is its lack of semantic information. As presented in Chapter 2, Web services rely exclusively on XML for interoperation. Unfortunately, XML guarantees only syntactic interoperability. Expressing message content in XML allows Web services parse each other's messages, but it does not provide any means on semantically "understanding" the message contents.

All current industry proposals for the Web services infrastructure are based on Web services reaching explicit agreement on both the way they interact and the format of the messages they exchange. Developers must hard code these interactions, as well as the how Web services should interpret the messages. Developers also need to modify their Web services when something changes in the interaction agreement or when something breaks. The fact that current Web services infrastructures do not support automatic Web service reconfiguration creates an infrastructure that is inflexible, brittle and expensive to maintain.

To overcome this brittleness, Web services need more autonomy which will let them reconfigure their interaction patterns as needed. Any increase in autonomy will let Web services react to potential changes in the environment while minimizing developer's direct intervention. The lack of explicit semantics prevents current Web services from acting autonomously. Autonomy will allow services to understand each other's messages as well as what tasks each service performs.

Autonomy can be achieved by using DAML-S which is both a language and ontology for describing Web services (Martin, 2003). DAML-S attempts to close the gap between the Semantic Web and current Web services.

Back to the booking flight example, an autonomous Web service would seek help from other services (cooperation) by itself without human request. If implemented properly, an autonomous Web service can provide personalized information to the client and constantly pursue the best flight ticket deal until the flight is booked.

Context awareness. We humans are context-aware. We are able to use implicit situational information, or context, to increase our conversational bandwidth (Brown, Davies, Smith and Steggles, 1999). This ability allows us to act in advance and anticipate other's needs.

Let us consider an example with two people located in the same room: Person A and Person B. Person A asks Person B, “close the door please.” As expected, Person B would instantly reason that Person A is referring to the door in the same room to be closed, and not the door in any other room.

This simple example illustrates context-awareness in human beings. The fact that Person B is able to respond and takes the right action is due to the following three valuable capabilities of humans:

1. Ontology sharing - humans are able to share communication languages and vocabularies.
2. Sensing - humans are able to perceive their environment through sensory organs.
3. Reasoning - humans are able to make sense out of what they have perceived based on the knowledge that they posses.

Let us assume that for some reason Person B is unable to share ontology with Person A or unable to sense from Person A or unable to make sense out of what he/she has perceived, then Person B would not be able to close the door as requested by Person A. This means that Person B becomes context-aware only when he/she possesses all of the three capabilities described above.

Ontology sharing, sensing and reasoning are not only crucial to human context-awareness, but also significant to the realization of context-aware applications. Because current Web services do not posses all three capabilities they cannot offer context-awareness.

CoolAgent RS is a multi-agent system that can automatically recommend different types of tailored information to users by reasoning from their context without any explicit manual input (Chen, Tolia, Sayers, Finin and Joshi, 2001).

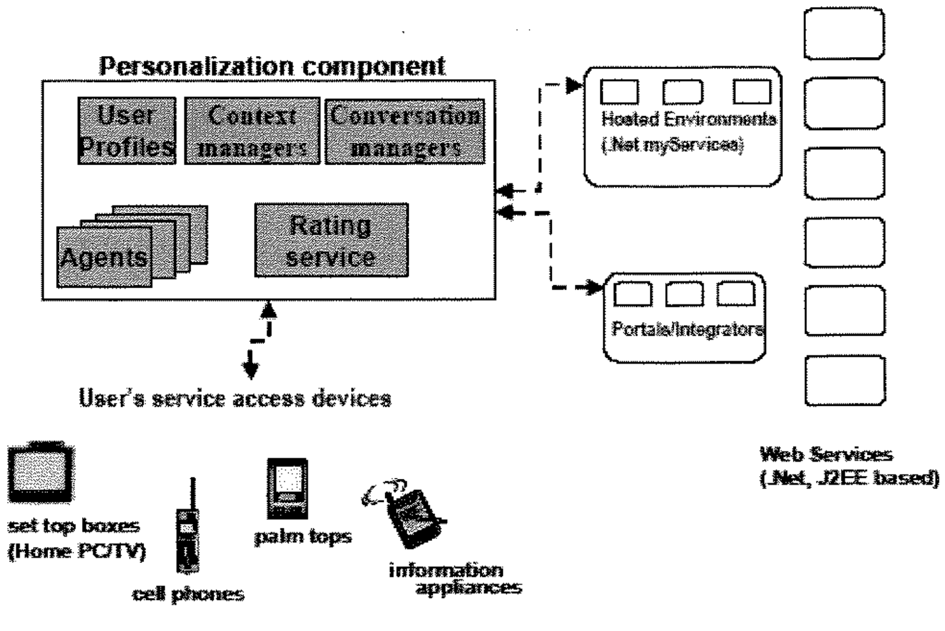
If our flight booking Web service would have been context-aware it would automatically behave differently and offer different information based on the fact if the flight was booked or not. It will also be able to notify the client if for any reason the flight is postponed or cancelled and offer different choices.

Personalized services/dynamic adaptation. Personalization describes the process of using customer information in order to optimize business relationship with them. Traditionally, personalization technology has focused on a service's interactions with the customer. The main goal of this technology has been to use information about a customer so as to better serve the customer by anticipating their needs, make the interaction efficient and satisfying for both parties, and build a relationship that encourages the customer to return for subsequent purchases.

Web services promise a lot of potential in providing personalization which is tightly linked with dynamic adaptation and context-awareness. The problem is that current standards on which Web services are based on do not support any type of personalization. This means that there is no way that a Web service can be aware of who is invoking it or keep track about client who invoked it in the past.

Research community has proposed several personalized service architectures that involve agent technology. For example, Kuno and Sahai came up with an architecture that uses personalization component to provide interaction between user devices and web services as shown in Figure 9 (Kuno & Sahai, 2002).

Figure 9. User devices interact with Web services via the personalization component


Personalization component
User Profiles
Context managers
Conversation managers
Agents
Rating service
Hosted Environments (.Net myServices)
Portals/Integrators
User's service access devices
set top boxes (Home PC/TV)
cell phones
palm tops
information appliances
Web Services (.Net, J2EE based)

The personalization component can be either integrated with hosted environments like .Net based services or with existing portals/integrators so as to provide user personalization. In this proposed architecture agents are used to implement delegates. These agents proxy for the users, achieve goals of the user through goal directed behaviour (user presence is optional) and also interact with other constituents of the personalization component.

For example, context managers track the user as its temporal and spatial location changes. Similarly, user profiles maintain static and dynamic information about the user. Static information is pre-registered while dynamic information is learnt by studying user behavioural patterns. As the delegates interact with Web services over time their experiences are recorded in the rating services. These rating services can be used by delegates of other users or can also be passed to third party rating services.

Overview of the research process

The organization and research process for this essay went through four main phases:

- Creating an outline
- Searching for relevant publications
- Categorizing and classifying resources
- Filtering resources

The following sections will describe these phases briefly.

Creating an outline

The outline creation went through three major changes. The suggested structure from the Style Guide was followed and modified as necessary. Considerable time was spent on planning the layout of the essay, the order in which the content should be presented and what depth should be used for certain areas. Suggestions from supervisor were also very helpful. After finalizing the outline of the essay it was much easier to search for, filter, and select research publications.

Searching for relevant publications. Depending on the chapter, the effort for finding relevant publications varied. ACM digital library, IEEE, CiteSeer, Google, and published books were used as sources of information. Some of the articles were not available for free in IEEE but they were published on CiteSeer or available online by authors themselves. Over one hundred different keywords and combinations of keywords were used to perform a fine search on relevant publications. More then thirty books were considered as sources but very few used because not much is published in this area.

The outcome of this searching process was a long list of references which later needed to be categorized, classified and filtered. At this stage, in order to compile a list of potential references, only a brief overview of articles was performed because of a relatively large amount of references.

Categorizing and classifying resources. After searching for relevant publications in the area of interest, all the publications were carefully categorized first by chapter that they might be used and then by the relevance to the topic. After that all the publications were classified into two large groups: short and long publications. During this phase most of the publications were carefully read in order to create an understanding of the state of the art in this field. This phase was the most consuming of the whole research process mainly because of the large number of articles and amount of reading that had to be done in order to properly address the issues.

During this process there were articles that were somewhat conflicting with each other because they were proposing solutions that use totally different technologies and implementations. In these cases more resources had to be found and proper conclusions were drawn based on architectures and technologies that are "mainstream" in this area of research.

Filtering resources by relevance. The final stage of the research process was the filtering of resources based on content and relevance of material. Some of the publications were custom tailored to solve a localized industrial problem but nevertheless they were taken into consideration because they showed the applied side of current technologies. While filtering resources there was sometimes a need to go back to phase one in order to verify the source of the article or even to search for more accurate articles.

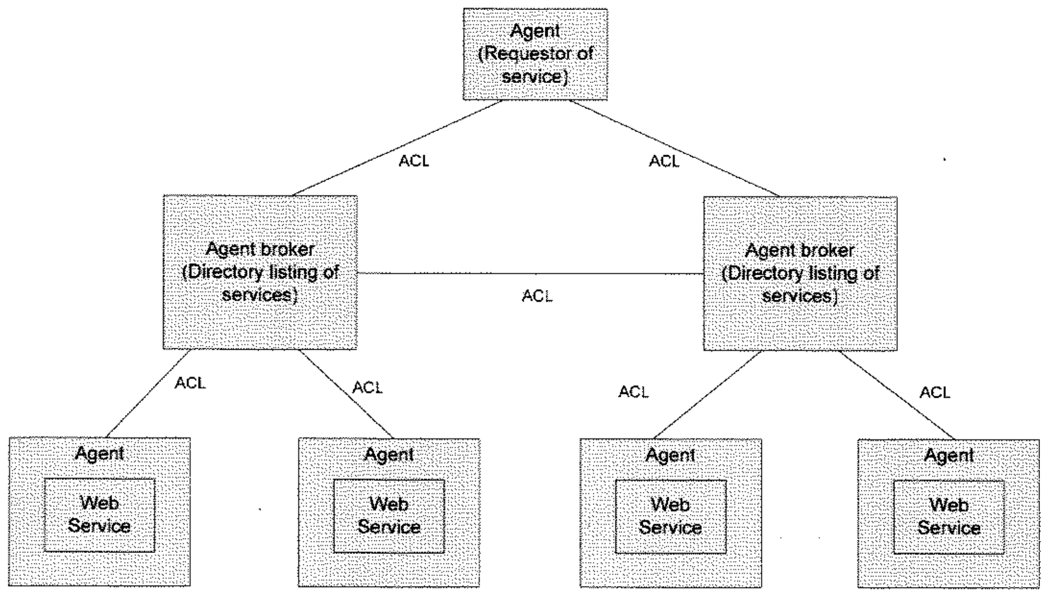
The four phase approach shown here was very helpful in gathering, organizing and filtering all the articles because it helped building the reference table from the beginning and using it as needed.

Findings

This section provides a more detailed summary of my personal findings during the period of working on this essay. I see Web services being the future of distributed computing which will allow integration of different platforms and architectures so they can connect seamlessly to each other. A lot of work has been done by leading software companies to standardize the protocols and languages on which Web services are based on. This is an ongoing process and yet a lot has to be done. On the other hand, agent technology, although a much more mature subject, is still struggling on the standardization aspect. Main reason for this is the lack of buy-in from major software companies to incorporate the technology into their products. There are only few stable commercial tools that offer agent-oriented development. Incorporating agent technology in Web services will open doors to a new era in distributed computing. Figure 10 presents a general view of my proposed architecture which is based on agent wrappers and will provide all the extra functionality that agents are known for. This means that agents will act as an extra layer of abstraction on top of Web services.

As Figure 10 illustrates, an interested party requests a service through an agent which in turn browses directory listings provided by an agent broker. Agent broker can communicate with other similar brokers to obtain other services not available in its directory. Services themselves can cooperate with other agents in autonomous mode depending on the level of autonomy either through an agent broker or directly (direct communication between agents is not shown). All the communication is done using ACL which simplifies the programming side of the implementation.

Figure 10. Proposed architecture for agent based Web services


Agent (Requestor of service)
ACL
ACL
Agent broker (Directory listing of services)
ACL
Agent broker (Directory listing of services)
ACL
ACL
ACL
ACL
Agent
Web Service
Agent
Web Service
Agent
Web Service
Agent
Web Service

The following sections will describe in more detail how the proposed agent architecture will support the aspects of cooperation, autonomy, context awareness and personalized services/dynamic adaptation.

Support for cooperation. As explained in previous sections existing Web services cannot cooperate among each other without a manual intervention of humans mainly because the existing communication protocols are not semantically rich. Using DAML-S as a communication language between agent-based Web services allows for smart communication between agents which in other words means agents "can understand" each other and automatically invoke services of other agents as needed. Based on the architecture presented in Figure 10 when the agent requestor searches for the needed service three scenarios can occur:

1. The found service is exactly what the client wanted (this is determined using a semantically rich communication language like DAML-S) and the agent requestor fulfills the client's request.
2. The service cannot be found but the agent (requestor of service) automatically is requests composition of services which involves cooperation of two or more services in order to meet the client's request.
3. The service cannot be completed in any way possible at the time requested. The client is notified of this condition and the service requestor continues to proactively search for service providers that can fulfill the clients request.

It is possible to modify the execution of the second scenario if the architecture presented in Figure 10 is modified to allow for a higher degree of autonomy of agent based Web services. This means is that the agent wrapper can have a higher level of autonomy which allows it to freely cooperate and communicate with other agents in order to fulfill the request from service requestor agent. This level of communication between agents is not presented in Figure 10 but it is worth mentioning because it directly affects the cooperation mode between agents.

Support for autonomy. Because of the fact that my proposed architecture uses only one language (protocol) to communicate between the requestor of the service and the corresponding provider, all involved parties are implemented using agent technologies so they can all communicate using the same language. DAML-S seems as an excellent candidate to use as a communication language between agents since DAML-S is both a communication language and ontology for describing Web services. Requestor of service is an agent capable of acting independently on behalf of its client. DAML-S allows agents to be autonomous because of its support for Semantic Web. This means that agents can make sense of the information because the information can be as semantically rich as needed. This allows agents to interpret the information on their own and make a decision on what to do next based on what is best for their clients. Therefore my proposed architecture inherently supports autonomy.

Support for context awareness. In previous sections of this chapter I have talked about CoolAgent RS which is a multi-agent system capable to automatically recommend different types of tailored information to users by reasoning from their context without any explicit manual input from users. This means that it is possible to adapt this system and add context awareness to the proposed architecture. The implementation and adaptation of the proposed architecture for context awareness is an undertaking that is beyond the scope of this essay but it makes a good candidate for future research in this field.

Personalized services/dynamic adaptation. As mentioned in previous sections of this chapter, research community has already proposed several personalized service architectures that involve agent technology. Figure 9 shows one of the implementations that uses personalization component to provide interaction between user devices and web services. Because agents are already part of my proposed architecture it is easy to integrate them with any of personalized service architecture proposed by research community. On the other hand there is also a possibility of further research on implementing personalization and dynamic adaptation on the proposed architecture. One more reason for this is the fact that context awareness is very tightly coupled with personalization and dynamic adaptation so all the features can be implemented together.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

Current Web services, without any support from agents, still do provide the capability to seamlessly integrate different platforms. They provide an excellent choice for implementing distributed applications because they are architecturally neutral. Web services operate using open, text-based standards, such as WSDL, UDDI, and SOAP, which enable components written in different languages and for different platforms to communicate.

Agent technology provides several advantages which can be easily incorporated in existing Web services. The capability of agents to be autonomous, cooperate with other agents, be aware of the context in which they are invoked, and dynamically adapt to changes in the environment are some of the main advantages that agents have compared to current Web services. Agent based Web services would provide clients with a fast, personalized, and intelligent service. This in turn will increase the percentage of returned customers because of higher satisfaction from services provided.

Instead of focusing on Web services separately from the agent technology, both technologies should be considered together. This approach makes it simpler to implement standards that organizations will have to follow which will lead to services that can be provided and consumed seamlessly.

There are several proposed architectures related to agent based Web services but none of them addresses all the possible advantages that agents have compared to existing Web services. Also, a lot more work needs to be done to get the industry buy-in because that is the only way we could finally see practical use of agent based Web services. For example, existing Web services are maturing as a technology mainly because major software companies are supporting them based on predefined standards. Many organizations have already published their services online to be consumed by customers, but these services are stateless, non-autonomous, lacking pro-activeness, in other words they are static services waiting to be consumed and that is it.

Further research is recommended on practical implementation and testing of the architecture provided in this essay. A proper agent communication language like DAML-S should be chosen and standardized as a communication language between agent-based services. I would also propose that a new name should be adopted for agent-based Web services to denote their new capabilities and make them visible as a new technology. For example, Intelligent Web Services (IWS) or Agent Based Web Services (ABWS) would be a much more appropriate name.

The proposed architecture comes as a result of a research of numerous related publications in this area. It is certainly a simple architecture which allows flexibility on implementation and deployment. Although not shown in Figure 10 this is a distributed architecture with three tiers involved. The first tier is the client tier which is represented with a “requestor of service” agent. The second tier is the agent broker which lists all the available services registered with that broker. The third tier is the service tier which represents the agent based services themselves. The distributed nature of the proposed architecture as well as single communication language (XML based) used between tiers inherently provides transparency, fault-tolerance (a must have feature of all distributed systems), extensibility and portability.

I strongly believe that the work presented in this essay contributes considerably towards the industry acceptance of Web services and more importantly agent based Web services. The essay also proposes a simple architecture that can be implemented using a programming language like Java, which is work that can be done in the future.

Further research is also suggested on justifying the use of a specific agent communication language (ACL) for agent based Web services. Somehow this ACL needs to be standardized as an official communication language between agent based Web services. Further research is recommended in integrating context awareness and personalization in the proposed architecture. As mentioned earlier it would be better if both these features were tackled at the same time because of their similarities.

Finally, more work has to be done to educate business owners on advantages of having services that their company offers exposed and easily accessible online. Many businesses are still in the process of implementing a web presence and they are unaware of Web services and what they can do for their business. Of course it would be useless to start and push for agent based Web services if most companies out there have no idea of what Web services are. I am referring here to business owners and not software vendors. It is useless to produce an item when there is a relatively small customer base interested in that item, mainly because the customer base does not know what to do with that item or how it can assist them in day to day life. To make things even more complicated, Web services are very young technology which just recently has been adopted and supported by major software vendors. Standards related to Web services are still under development which alerts many companies on rather waiting and doing nothing than adopting the new technology.

REFERENCES

Benfield, S. (2001). Web Services: The Power to Change the World? *Web Services Journal*, June, pp. 22.

Berners-Lee, T. (April 2001). Services and Semantics: Web Architecture. Retrieved December 27, 2004, from http://www.w3.org/2001/04/30-tbl.

Booch, G. (1994). *Object-Oriented Analysis and Design* (2nd ed.). Addison-Wesley: Reading, MA, pp.91.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. & Orchard, D. Web Services Architecture. W3C, Working Draft http://www.w3.org/TR/2003/WD-wsarch-20030808/, 2003.

Borck, J. (2002). InfoWorld Technology of the Year: Web Services. *InfoWorld*, February, pp. 48.

Brazier, F. M. T., Dunin-Keplicz, B. M., Jennings, N. R., & Treur, J. (1996). DESIRE: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, *6*, 1, pp.67-94.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, & J., Perini, A. (2002). *Tropos: An agent-oriented software development methodology*. Technical Report DIT-02-0015, University of Trento, Department of Information and Communication Technology.

Brown, P., J., Davies, N., Smith, M., Steggles, P. (September 1999). Towards a better understanding of context and context-awareness. Handheld and ubiquitous computing, number 1707 in Lecture Notes in Computer Science, pp. 304-7.

Buhler, P. A. & Vidal, J. M. (2003). Semantic Web Services as Agent Behaviors. In B. Burg, J. Dale, T. Finin, H. Nakashima, L. Padgham, C. Sierra, and S. Willmott, Agentcities: *Challenges in Open Agent Environment*, Springer-Verlag, pp.25-31.

Burrafato, P. & Cossentino, M. (27-28 May 2002). Designing a multi-agent solution for a bookstore with the PASSI methodology. *In Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*: Toronto, ON., Canada, at CAiSE'02.

Bush, G., Cranefield, S., & Purvis, M. (January 2001). *The Styx agent methodology*. The Information Science Discussion Paper Series 2001/02, Department of Information Science, University of Otago, New Zealand. Retrieved July 24, 2004, from http://divcom.otago.ac.nz/infosci.

Caire, G. & Leal, F. (July 2001). *Recommendations on supporting tools*. Technical Information Final version, European Institute for Research and Strategic Studies in Telecommunications (EURESCOM), Project pp.907, deliverable 4.

Casati, F., Ilnicki S., Jin, L. J., Krishnamoorthy, V. and Shan M. C. (2000). eFlow: a Platform for Developing and Managing Composite e-Services. Technical Report HPL-2000-36, HP Laboratories, Palo Alto, California, USA.

Cauldwell, P., Chawla R., Chopra, V., Damschen, G., Dix, D. and Hong, T. (2001). *Professional XML Web Services*. Birmingham, UK: Wrox Press, pp. 21.

Chen, H., Tolia, S., Sayers, C., Finin, T. and Joshi, A. (September, 2001). Creating context-Aware Software Agents. Retrieved January 12, 2005, from http://ebiquity.umbc.edu/v2.1/_file_directory_/papers/80.pdf.

Collinot, A., Drogoul, A., & Benhamou, P. (December 1996). *Agent oriented design of a soccer robot team*. In Proceedings of the Second International Conference on Multi-Agent Systems, Kyoto, Japan.

Debenham, J. & Henderson-Sellers, B. (May 2002). Full lifecycle methodologies for agent-oriented systems - the extended OPEN process framework. *In Proceedings of Agent-Oriented Information Systems (AOIS-2002) at CAiSE'02*, Toronto.

DeLoach, S. A., (May 1998). Multiagent systems engineering: A methodology and language for designing agent systems. *In Agent-Oriented Information Systems '99 (AOIS'99)*, Seattle WA.

FIPA, (Oct. 2002). FIPA Communicative Act Library Specification. Technical Report XC00037I, Foundation for Intelligent Physical Agents.

Florescu, D., Grunhagen, A. & Kossmann, D. (May 2002). XL: An XML Programming Language for Web Service Specification and Composition. In Proceedings of the International World Wide Web Conference, pages 65-76, Honolulu, Hawaii, USA.

Foster, I., Kesselman, C., Nick, J.M. & Tuecke, S. (2002). Grid Services for Distributed Systems Integration. IEEE Computer, 35 (6). 37-46. 2002.

Glaser, N. (1996). *Contribution to knowledge modelling in a multi-agent framework (the CoMoMAS approach)*. PhD Thesis, L'Universite Henri Poincare.

Goble, C.A., De Roure, D., Shadbolt, N.R. & Fernandes, A. (2004). Enhancing Services and Applications with Knowledge and Semantics. The Grid: Blueprint for a New Computing Infrastructure (2nd Edition), Morgan-Kaufmann.

Guarino, N. & Giaretta, P. (1995). Ontologies and knowledge bases: Towards a terminological clarification. In N. Mars, editor, Towards Very Large Knowledge Bases. IOS Press.

Howerton, B. (2002). Keeping Promises. *Intelligent Enterprise*, January, pp.46.

Huhns, M. N. (2002). Agents as Web Services. *IEEE Internet Computing*, July-August, pp.93-94.

Iglesias, C. A., Garijo, M., Gonzlez, J. C., & Velasco, J.R. (1996). A methodological proposal for multiagent systems development extending CommonKADS. *In Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop.*

Ishikawa, F., Yoshioka, N., & Tahara, Y. (2004). Toward Synthesis of Web Services and Mobile Agents. Presented in Web Services and Agent-Based Engineering (WSABE) workshop. Retrieved August 15, 2004, from http://www.agentus.com/WSABE2004/program/.

Jennings, N. R., Faratin, P., Norman, T. J., O'Brien, P., Odgers, B., & Alty, J. L. (2000). Implementing a business process management system using ADEPT: A real-world case study. *International Journal of Applied Artificial Intelligence.*

Kendall, E. A., Malkoun, M. T., Jiang, C. H. (1995). *A methodology for developing* *agent based systems*. In Chengqi Zhang and Dickson Lukose, editors, *First Australian Workshop on Distributed Artificial Intelligence*.

Kinny, D. & Georgeff, M. (1996). *Modelling and design of multi-agent systems. In Intelligent Agents III: Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*. LNAI 1193. Springer-Verlag.

Korzeniowski, P. (2002). A Little Slice of the UDDI Pie. eWeek, 4 February, pp.50-51.

Kuno, H., Sahai, A., (2002). My Agent Wants to Talk to Your Service: Personalizing Web Services through Agents. Hewlett-Packard Laboratories, April 2002.

Lazcano, A., Schuldt, H., Alonso G. & Schek, H. J. (March 2001). WISE: Process based E-Commerce. IEEE Data Engineering Bulletin, 24(1):46-51.

Lind, J. (1999). ASSIVE: Software Engineering for Multiagent Systems. *PhD thesis*, University of Saarbrcken, Germany.

Ludwig, H. & Hoffner, Y. (February 1999). Contract-based Cross-Organisational Workflows - The CrossFlow Project. In Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration, pages 1-6, San Francisco, California, USA.

Martin, D. (2003). DAML-S: Semantic Markup for Web Services. Retrieved January 10, 2005, from http://www.daml-s.org/daml-s/0.9/daml-s.html.

Mayfield, J., Labrou, Y., & Finin, T. (1996). *Evaluation of KQML as an Agent Communication Language*. Retrieved July 26, 2004, from http://www.cs.umbc.edu/~jklabrou/publications/lnai95.pdf.

Medjahed, B. (January 2004). Semantic Web Enabled Composition of Web Services. Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science and Applications.

Medjahed, B., Bouguettaya, A. & Elmagarmid, A. (November 2003). Composing Web Services on the Semantic Web. The VLDB Journal, 12(*4*):333-351.

McGarr, M. (2002). Transforming Business Processes with EDI. *Electronic Commerce World*, May, pp.23.

Newcomer, E., (2002). Understanding Web Services. XML, WSDL, SOAP, and UDDI. Boston, MA. Pearson Education.

Object Management Group. (1999). Unified Modeling Language Specification (Version 1.3). Retrieved December 28, 2004, from http://www.omg.org/technology/documents/formal/uml.htm.

Odell J., Parunak, H. V. D., & Bauer B. (2000). *Representing Agent Interaction Protocols in UML*. The First International Workshop on Agent-Oriented Software Engineering (AOSE-2000).

Padgham, L. & Winikoff, M. (July 2002). Prometheus: A methodology for developing intelligent agents. *In Third International Workshop on Agent-Oriented Software Engineering.*

Ponnekanti, S.H. & Fox, A. (2002). *SWORD: A Developer Toolkit for Web Service Composition*. In Proceedings of 11th WWW Conference, Honolulu, Hawaii.

Ramchurn, S.D., Huynh, D. & Jennings, N.R. (2004). Trust in Multiagent Systems. The Knowledge Engineering Review.

Schuster, H., Baker, D., Cichocki, A., Georgakopoulos, D. & Rusinkiewicz, M. (March 2000). The Collaboration Management Infrastructure. In Proceddings of the IEEE International Conference on Data Engineering, pages 485-487, San Jose, California, USA.

Searle, J. (1969). Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press.

Shen, M., Benatallah, B., Dumas, M. & Mak, E., O., Y. (August 2002). SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. In Proceedings of the International Conference on Very Large Databases, pages 1051-1054, Hong Kong, China.

Sun Microsystems, (2005). Java Web Services. Java Web Services Developer Pack (Java WSDP). Retrieved January 15, 2005, from http://java.sun.com/webservices/jwsdp/index.jsp.

Wagner, G. (July 15 2002). *A uml profile for external aor models*. In Springer-Verlag LNAI, editor, In Proceedings of Third International Workshop on Agent-Oriented Software Engineering (AOSE-2002), held at Autonomous Agents and Multi-Agent Systems (AAMAS 2002), Bologna, Italy.

Weiss, G. (January 2002). *Agent Orientation in Software Engineering*. Revised Version for Knowledge Engineering Review. Retrieved July 27, 2004, from http://wwwbrauer.in.tum.de/~weissg/Docs/weissg-ker02.pdf.

Weissenfels, J., Gillmann, M., Roth, O., Shegalov, G. & Wonner, W. (February 2000). The Mentor-Lite Prototype: A Light-Weight Workflow Management

System. In Proceedings of the IEEE International Conference on Data Engineering, pages 685-686, San Diego, California, USA, February 2000.

Wilson, T., (2001). UDDI Promises Link to Web Services. Internet Week, 26 November, pp.26.

Wooldridge, M. & Ciancarini, P. (2000). *Agent-Oriented Software Engineering: The State of the Art*. Retrieved July 22, 2004, from http://www.csc.liv.ac.uk/~mjw/pubs/aose2000a.pdf.

Wooldridge, M., Jennings, N.R., & Kinny, D. (1999). A methodology for agent-oriented analysis and design. *In Proceedings of the third international conference on Autonomous Agents* (Agents-99).

World Wide Web Consortium. (2004). Extensible Markup Language (XML). Retrieved June 2004, from http://www.w3.org/XML/.

World Wide Web Consortium. (2004). Web Services Description Working Group. Retrieved July 2004, from http://www.w3.org/2002/ws/desc/.

Yang, J. & Papazoglou, M. (2002). *Web Component: A Substrate for Web Service Reuse and Compositio*. In Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE02), May, Toronto, LNCS, Vol. 2348, p21-36.

Zou, Y., Finin, T., Peng, Y., Joshi, A., Cost, S. (2003). Agent Communication In DAML World. University of Maryland. Retrieved January 10, 2005, from http://www.cs.umbc.edu/~yzou1/publications/wracbook.pdf#search='Agent%20Communication%20in%20DAML%20World'.

APPENDIX

Table 1 Agent-Oriented Software Engineering methodologies.

Methodology	**Authors**	**Category**	**References**
AAII	Kinney, Georgeff & Rao	KE	Kinney et. al., (1996)
ADEPT	Jennings et. al.	KE	Jennings et. al., (2000)
AO methodology for enterprise modeling	Kendall et. al.	OO	Kendall et. al., (1995)
AOR	Wagner	OO	Wagner, (2002)
Agent UML	Odell et. al.	OO	Odell et. al., (2000)
Cassiopeia	Collinot et. al.	OO	Collinot et. al., (1996)
CoMoMAS	Glaser	KE	Glaser, (1996)
DESIRE	Brazier et. al.	OO	Brazier et. al., (1997)
Gaia	Wooldridge, Jennings & Kinny	OO	Wooldridge et. al., (1999)
OPEN agents	Debenham & Henderson	OO	Debenham et. al., (2002)
MaSE	DeLoach	OO	DeLoach, (1998)
MAS-CommonKADS	Iglesias et. al.	KE	Iglesias et. al., (1996)
MASSIVE	Lind	OO	Lind, (1999)
MESSAGE	Caire et. al.	OO	Caire et. al., (2001)
PASSI	Burrafato et. al.	OO	Burrafato et. al., (2002)
Prometheus	Padgham & Winikoff	OO	Padgham et. al., (2002)
Styx	Bush et. al.	OO	Bush et. al., (2001)
Tropos	Bresciani et. al.	KE	Bresciani et. al., (2002)

Notation: "OO" is short for Object-Oriented and "KE" is short for Knowledge Engineering

Table 2 Web Service composition prototypes vs. Interaction Layers

Prototype	Communication Layer	Content Layer Business	Process Layer
CMI	Transport protocols (e.g., HTTP, CORBA) must a priori be agreed upon	Message format (e.g., XML, EDI) must a priori be agreed upon	State machine based model for process description
WISE	Coordination and communication module	Not Addressed	Virtual business processes
CrossFlow	Java RMI	Contract in XML	Contracts
Mentor-Lite	Transaction Processing (TP) monitor (Tuxedo)	Not Addressed	Business processes expressed as state and activity charts
eFlow	Java RMI	Provides adapters to support different protocols such as OBI and RosettaNet	State machine based model for process description
XL	SOAP	XML Schema	Little or no statements for inter-service business processes
SELF-SERV	SOAP	Not Addressed	State charts

Table 3 Prototypes vs. Interaction Dimensions

Prototype	**Coupling**	**Autonomy**	**Heterogeneity**	**Ext. Manag.**	**Adaptability**	**Security**	**Scalability**
CMI	Tight and long term	External systems only need to reveal the state they are in after they accomplish a task, not how they accomplish the task	Use of object-oriented proxies	State dependent control flow and use of awareness events	Primitives such as *optional* and *inhibitor* can be used for coping with some unforeseen events	Role	Distributed and parallel engines for execution
WISE	Tight and long term	Partners must advertise services in encapsulated objects	Object-based middleware	Process monitoring and analysis module	Execution guarantee	Not Ad-dressed	Distributed architecture
Cross Flow	Loose and transient	Partners must agree on service contract definition	Partners must install service contract run time environment	Quality of Service (QoS) module provides monitoring facilities	Primitives for flexible execution are restricted to those provided by traditional workflows	Not Addressed	Cost of entry: participants must locally install contract run time environment
Mentor-Lite	Tight and long term	Participants do not need to reveal how services are implemented	Application programs are connected to the workflow engine by specific wrappers	Not Ad-dressed	Not Addressed	Not Ad-dressed	Workflows are partitioned into several sub-workflows and distributed

eFlow	Loose and long term	External systems need to describe their services not their implementation	Provides adapters for different protocols and platforms such as OBI, RosettaNet, and e-speak	Event tracking	Provides process templates, service nodes, and service data repositories for reuse	Not Addressed	Distributed service enactment engines
XL	Loose and transient	Participants do not need to reveal how services are implemented	Web services can be written in XL, Java, or other languages	History clauses	Change clauses	Security features of J2EE	Not Addressed
SELF-SERV	Loose and transient	Participants do not need to reveal how services are implemented	Service wrappers	Not Addressed	Not Addressed	Not Addressed	Peer-to-peer execution model