www.icgst.com

A Mobile Agent Framework–Based Metadata Representation

Yasser K. Ali[1], Hesham N. Elmahdy[1], S. H. Ahmed[2]
[1]Information Technology Department,
[2]Vice Dean for Education and Students Affair,
Faculty of Computers and Information, Cairo University

Abstract

Mobile agents can significantly improve the design and the development of Internet applications. Mobile agents have characteristics of autonomy and adaptability to open and distributed environments. This paper proposes mobile agents framework. This framework presents a mobile agent system design based on metadata representation .we introduce the advantages of mobile agents in implementing web services, which have all characteristics of distrusted systems.

Keywords*:* Distributed Systems, Mobile Agent, Web Services, Extensible Markup Language (XML), Metadata.

1. Introduction

As information becomes distributed across wide areas, information management becomes an issue to be addressed. Due to the problems of latency, intermittent connectivity and variable service availability, it is difficult to ensure that consistency updates are made in a timely fashion [1]. "A software agent can be defined as a software entity that functions continuously and independently in a particular environment, it is able to carry out activities in a flexible and intelligent manner that is responsive to changes in the environment"[2]. "An **autonomous agent** (object) can be programmed to satisfy one or more goals, even if the agent (object) moves and loses contact with the creator. A **mobile agent** (object) has the ability to move independently from one device to another on a network. Mobile agents are generally serializable and persistent".[3]

Mobile agents are programs that encapsulate data and code, which may be dispatched from a client computer and transported to a remote server for execution. Mobile agents execute asynchronously and autonomously. Once a user has created an agent, it can run without intervention from the user. The agent performs its task and saves any results until its connection to the user is re-established. Mobile agent provides a reliable transport between a client and server without necessitating a reliable underlying communications medium. [3]

This paper focuses on mobile agent framework, which gives support for building web applications transparent and platform independent components. The contribution of this paper is to propose a model of mobile agent framework; which consists of a hierarchy of classes (structured as a Java package) and a visual developing tool. Where: Mobile agents are annotated with metadata to describe services and agent life-time is bound to a service request

This paper delineates this subject in five sections after the introduction section. The second section explains fundamentals of agents and web services techniques and related tools such as XML, metadata, and communication protocols. The third section provides related work in mobile agent and web services integration. The fourth section describes the framework architectures and its fundamental blocks such as agents, hosts, and services. The fifth section presents our implementation prototypes and scenario outline are provided. Finally, conclusion and future work are presented in section six.

2. Background

2.1 Intelligent Mobile Agent

We can consider any mobile agent to be an intelligent mobile agent if [3].

- If a task must be performed independently of the computer that launches the task, a mobile agent can be created to perform this task. Once constructed, the agent can move into the network and complete the task in a remote program.

- If a program needs to send a large number of messages to objects in remote programs, an agent can be constructed to visit each program in turn and send the messages locally. Local messages are much faster than remote messages.
- If you want to partition your programs to execute in parallel, you can distribute the processing to several agents, which migrate to remote programs and communicate with each other to achieve the final goal.
- If periodic monitoring of a remote object is required, creating an agent that moves to the remote object and monitors it locally is more efficient that monitoring the object across the network.
- If a series of operations must be performed inside a portable device such as a cell phone or pda that is only occasionally connected to a network then an agent can move into the device, perform its task, and move back into the network only when necessary.

2.2 Web Service Technology

Web service is a software system designed to support interoperable machine-to-machine interaction over a network [4]. It has an interface described in a machine-processable format - specifically "Web Services Description Language" (WSDL) [5]. Other systems interact with the Web service in a manner prescribed by its description using "Simple Object Access Protocol" (SOAP) messages [6], typically conveyed using Hyper Text Transfer Protocol (HTTP) [7] with an XML serialization in conjunction with other Web-related standards [8]. This definition should give a vision what web services really are. It's describing a collection of protocols and open standards for exchanging data between software applications written in various programming languages and running on various platforms. A web service includes the following technologies: WSDL, SOAP and "Universal Description, Discovery, and Integration" (UDDI) [9] .WSDL is the standard means for expressing Web service descriptions. SOAP is a protocol defining the exchange of messages containing Web service requests and responses. UDDI is the directory services schema commonly used to register and discover Web services as shown in figure 1.

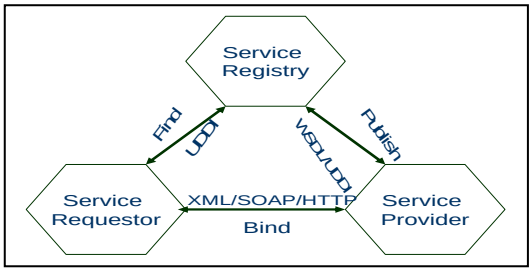

Figure 1.Web Services Basic Components

2.3 Metadata

Metadata is structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource. Metadata is often called data about data or information about information [10]. The term metadata is used differently in different communities. Some use it to refer to machine understandable information, while others use it only for records that describe electronic resources. In the library environment, metadata is commonly used for any formal scheme of resource description, applying to any type of object, digital or non-digital, Table 1 describes all types of metadata.

There are three main types of metadata:

• Descriptive metadata describes a resource for purposes such as discovery and identification. It can include elements such as title, abstract, author, and keywords.

• Structural metadata indicates how compound objects are put together, for example, how pages are ordered to form chapters.

• Administrative metadata provides information to help manage a resource, such as when and how it was created, file type and other technical information, and who can access it. There are several subsets of links to resources based on audience or topic. Such lists can be built as static WebPages, with the names and locations of the resources "hard coded" in the "Hyper Text Markup Language" (HTML). However, it is more efficient and increasingly more common to build these pages dynamically from metadata stored in databases.

Table 1: Types of Metadata

Acronym	Name	Description
XML	Extensible Markup Language	Defines document content using metadata tags and namespaces
DTD	Document Type Definition	Defines XML document structure (analogous to DDL schema)
XSL	Extensible Style Language	XSL or Cascading Style Sheets (CSS) separate layout from data
XLL	Extensible Linking Language	XLL implements multi-directional links (single or multiple)
DOM	Document Object Model	Implements a standard API for processing XML in any language
RDF	Resource Description Framework	W3 Interoperability Project for data content interchange

2.4 XML and Metadata

Metadata is used to define the structure of an XML document or file. Metadata is published in a Document Type Definition (DTD) file for reference by other systems. A DTD file defines the structure of an XML file or document. It is analogous to the Database Definition Language (DDL) file that is used to define the structure of a database, but with a different syntax [11].

An example of an XML document identifying data retrieved from a PERSON database is illustrated in Figure **2**. It includes metadata markup tags (surrounded by < … >, such as <person_name>) that provide various details about a person. From this, we can see that it is easy to find specific contact information in <contact_details>, such as <email>, <phone>, <fax> and <mobile> numbers.

```
<PERSON person_id="p1100" sex="M">
        <person_name>
        <given_name>yasser</given_name>
        <surname>kamal</surname>
        </person_name>
    <collage>
    Computers and Information
    </collage>
    <country>Egypt</country>
    <contact_details>
        <email>yasser_kamal@hotmail.com
        </email>
        <phone>02-4096617</phone>
        <fax>02-40-8322</fax>
        <mobile>0101932430</mobile>
    </contact_details>
</PERSON>
```

Figure: 2 An Example of an XML Document with Metadata Tags

3. Related Work

Much researches have been conducted on services integration. Some of which utilize mobile agents. However, not of all combine the mobile agent technology with standards for web services integration. A few proposed metadata with both technologies.

Antonio Corradi, Rebecca Montanari, [12] has been proposed a framework for configurable semantic support to mobile users, called MASS (Middleware for Adaptive Semantic Support). MASS focuses on two peculiar aspects. Firstly, it exploits the visibility of two kinds of metadata, user/device profiles and policies, to tailor semantic support functionalities. This configuration feature enables the framework to adapt semantic functionalities to several kinds of users and devices, thus dealing with the heterogeneity typical of pervasive environments. Secondly, it allows each mobile device to exhibit its semantic functionalities, so that they can be accessed by users in the vicinity, and it enables the device to discover and to exploit semantic support capabilities offered by the nearby devices. D.G.A. Mobach, B.J. Overeinder, N.J.E. Wijngaards, and F.M.T. Brazier[13], have been proposed a management architecture, including a management oriented agent life cycle model for AgentScape,[14]AgentScape is designed to support and manage heterogeneous agents. To this purpose AgentScape's management system uses a management-oriented agent life cycle model to describe the state of heterogeneous agents. Within this model, the suspended state is viewed as the central state of an agent. Open questions that will be addressed in future research concern interoperability with life cycle models in other multi-agent system frameworks, as well as interoperability implications when multiple extensions of the life cycle model are used concurrently.

Dominic Greenwood and Monique Calisti [15] are identifying a means of connecting agents and Web services. Due to the evident technology mismatches between Web services and software agents, including strong vs. loose coupling and representational encodings, they have been identified an approach that introduces an intermediary service entity, the Gateway architecture for enabling transparent, automatic connectivity between Web services and agent services , which is designed to encapsulate the functionality required to connect the two domains, whilst ensuring minimal human intervention and service interruption.

Lyell et al. [16] has been discussed the concept of a hybrid "Java 2 Platform, Enterprise Edition/ Foundation for Intelligent Physical Agents" (J2EE/FIPA)-compliant software agent system using Colored Petri Nets. This approach identifies two key concepts that (1) agents should be able to expose their services as Web services for the potential use of non-agent clients and (2) these Web service-enabled agents should advertise using both a UDDI registry and a FIPA Directory Facilitator (DF). In general, our proposed architecture attempts to benefit from the desired properties that are inherent to web services and to the mobile agents, while overcoming the limitations that are related to each technology when employed alone.

The proposed solution concerns the use of mobile agents for web services dynamic discovery and integration, through the extensive use of mark up languages that relies the notation of distributed systems. The perspective framework main contribution lies in designing agents annotated with metadata to describe services, and its life time is bounded to a services request. More specifically, mobile agent represents the requester of the web services. and agent migrates to registry and discover the web services., moreover , mobile agents actions are described in it's policy repository.


4. Framework Architecture

4.1 Framework Fundamental Blocks

The fundamental blocks of an agent system are agents, hosts and services. An overview of the system is presented in figure 3. An agent is dispatched by a client to a host, where it operates in the execution environment of that host .The agent can, if needed, migrate between hosts and can make requests to different services provided by hosts. Invoked services return arbitrary information that can be processed by agents. The proposed framework does not impose any restriction regarding the implementation of needed services.

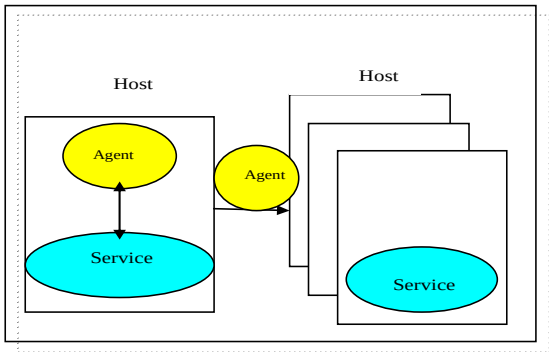

Figure: 3 The Fundamental Blocks of an Agent System

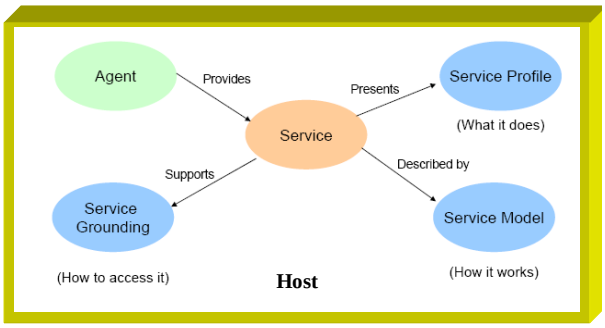

Figure: 4 Agent Execution Environments

The two main parts of an agent framework are, the agents and the agent hosts:

Agent: small piece of software, mostly aimed at solving a specific task on behalf of a human user (directly or indirectly) [17]. An agent could perform the given task on one or more agent hosts. Agents are not stand-alone programs, since they require a host to run.

Host: a server program that executes one or more agents. It provides a secure execution environment for the agent, which includes persistence, transactions and protection from other agents. The host receives the agents through the standard/proprietary migration process. It provides services that the agents can use. The host also manages the communication between the agents and provides the services that they need.

Naming service [18]: Naming generally involves assigning a location independent name to each agent. Since agents potentially migrate and clients or other entities (such as other agents) may need to locate them, there is a need for an agent-naming facility. This facility is based on the naming convention [19], and the naming service. Naming convention specifies what an agent's name is, and how it is used (like Aglets [20], our approach uses "Uniform Resource Identifier" (URIs) to name agents). The naming service is a generic interface to aid the use of the naming conventions. Both can be considered as abstraction levels; the naming convention is the low-level that specifies how to create names, obtain references and de-reference names into agents. The naming service is the high-level abstraction that makes use of the low-level design to simplify agent name de-referencing. Each core component of the framework is denoted by an abstract class in order to assure the system scalability and adaptability and to give developers the possibility to design any kind of agents and/or hosts.

4.1.1 Agent

The agent concept is encapsulated by the Agent abstract class. A task can be added to an agent, to be performed within the environment. The task is denoted by an abstract class in order to provide flexibility to the agent, as shown in figure 5. Thus, an agent can perform any task; it is not restricted only to some actions that it can perform. Of course, at the implementation level, the developer should describe the task as Java source-code.

Information regarding the agent can be obtained via an AgentMetaData class which provides data about its name, creator, owner, location etc. At the implementation level, metadata can be stored and processed as XML documents (for example, as "Resource Description Framework" (RDF) constructs [21]). In order to give support for the semantic Web applications, metadata can include information about the relationships between agents and other components (e.g., agents, hosts, services, users, etc).

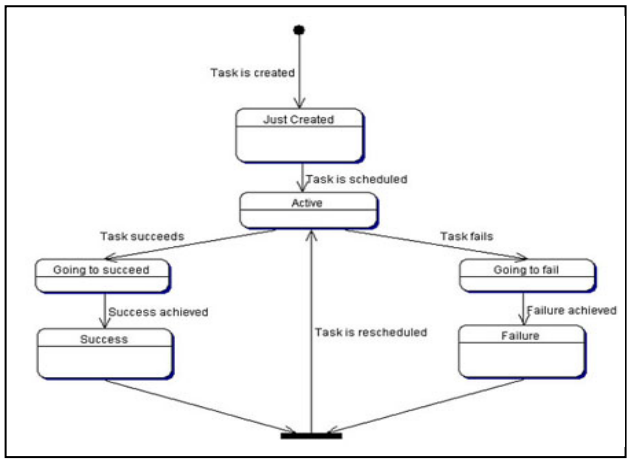

Figure 5: Task State Diagram

4.1.2 Host

The host is also encapsulated by an abstract class, based on a server abstraction The server should rely on client/server paradigm (at the implementation level, we used sockets), in order to communicate with clients. Another solution is to adopt the peer-to-peer model [22].

The structure of agent host uses different basic concepts, such as: Containment (a container for the hosted agents) [23], Service access (an interface to access the host services), Messaging support (provides communication between agents/hosts), Migration support (feature to support mobile agents).These components were implemented in abstract classes, which will run in parallel within the host context as shown in figure 6. Each such as abstract class is denoted by a Java service using events and listeners to communicate and launching exceptions to signal special conditions.

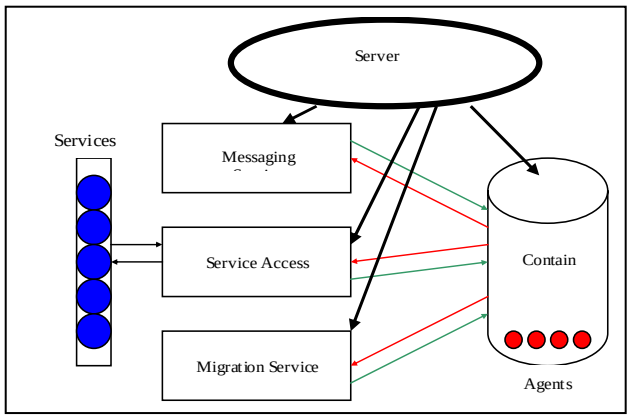

Figure 6: Agent Host

A naming service maps a name for an entity to a set of labelled properties. The Domain Name System (DNS) [24], for instance, is a network naming service that maps easy-to-remember names to hard-to-remember network addresses. Similarly, name services can be put to use in distributed computing architectures to map the identity of a mobile software agent to its current location on the network the names for the mobile agents must be:

• Unique: This is a requirement common to all naming services.

• Persistent: The names must not change unless the naming service is notified.

The mobile agents can either specify their home base directly or can have a home base assigned. The naming service recognizes and is capable of resolving two types of names: location-dependent names or location-independent names. Due to this versatility, the service can effectively shift from a location-dependent service to a location-independent service in the event of a node failure comprised of the following [25]:

1. Unique Names: Mobile agents have unique names that are premised upon Uniform Resource Unique Names: Mobile agents have unique names that are premised upon Uniform Resource Identifiers (URI) [26]. The URI can take the form of a URL [27], which is location-dependent and is how the agent specifies its home base, or the URI can take the form of a URN, which is location-independent and requires a home base to be assigned.

2. Home Bases: Nodes on the network that provide the naming service for the mobile agents.

3. Mapping Records: A record kept for a mobile agent that is kept current with the agent's location on the network. The record is stored at the agent's respective home base.

4.2 Mobile Agent and Web Services Integration

A web services should be able to invoke an agent services and vice versa To integrate mobile agent and web service technology in a seamless manner, components have to be designed, which map between the different mechanisms for service description, service invocation, and service discovery, in both worlds. In other words, messages representations from the according web service protocols (WSDL, SOAP, UDDI) have to be translated into corresponding requests data types of the agent system, and vice versa.

This section describes three issues that exist at the boundary of mobile agents and web services: service provision, request – agent mapping, and agent communication

4.2.1 Service Provision

Web Services Description Language (WSDL), describes services as operations on messages at a particular network end-point, and with bindings to concrete protocols and message formats, this methods call pure WS invocation from agent to agent or agent to services and vice verses, as shown in figure [4].

It would be possible to simply require the programmer to provide a WSDL description of any mobile agents [28]. An alternative method of service provision is a generic functionality of mobile agents that provide and revoke services by identifying a network endpoint and a bundle of methods that requests should be dispatched to. In this case the provider is represented by a stationary agent which communicates with the mobile agent that represents the client. The stationary agent invokes the web service on behalf of the user's mobile agent.

4.2.2 Web Services and Mobile Agent Mapping

When representing mobile agents as a java objects a Web service request maps to a particular mobile agent that fulfils that request.

Metadata in the framework environment is expressed according to the Resource Description Framework (RDF) . In essence, RDF is a format for describing semantic networks or directed graphs with labelled edges. Nodes and edges are named with


uniform resource identifiers (URIs), making them globally unique and thus useful in a distributed environment. Node URIs is used to represent objects, such as web pages, people, agents, and documents. A directed edge connecting two nodes expresses a relationship, given by the URI of the edge. A standard called RDF Schema. RDF Schema specifies a way for schema writers to define meanings for these edge URIs, which are called RDF properties. Because URIs are globally unique (like Java package names, typically URIs are generated to include an Internet domain name), the possibility of namespace conflict is negligible. A URI can be used as a "contract" since its use implies consistency with the semantics provided by the party defining the URI.

RDF in itself will offer an extensible type system which allows one to build class hierarchies and - due to RDF's expressive capabilities - can be used to specify ontologies or term vocabularies. This places RDF as a language which agents can use for describing their capabilities and negotiating the terminologies used in communication. The road to agent architectures requires description mechanisms, and RDF could be used in conjunction with other agent languages such as KQML [29] to handle complex representational tasks. In fact, in many cases RDF could be used to substitute KIF [30] as a more broadly understood in our approach we proposed RDF conjunction with "Agent Communication Language- Foundation for Intelligent Physical Agents" ACL-FIPA

We see the future of distributed object applications to be built using various multiagent techniques. It is essential, however, that agents are supported by strong mechanisms for describing not only the agents themselves and their capabilities, but also other resources on the web. Resource discovery by agents can enable qualitatively more flexible applications than those in existence today, due to the fact that systems can be built to intelligently react to situations and environment not known at the time of system design.

4.2.3 XML and Agent Communication

Instead of sending an RPC message from one object to another, an agent communication language (ACL) establishes an inter-agent communication protocol for exchanging information and coordinating multiple autonomous agents [31]. Software agents thus encapsulate a more seful, goal-oriented component, as opposed to the smaller units of functionality encapsulated by a typical business object. An ACL must similarly enable a more purposeful conversation among agents, not a basic, low-level message exchange.

4.2.3.1 FIPA ACL

The most important output of FIPA today is its Agent Communication Language (ACL), which is based on speech-act theory [32]. A FIPA ACL message consists of a header, the 'communicative act', followed by the subject of this act, referred to as the 'content'. FIPA ACL acts, such as 'inform', 'request', 'propose' or 'cfp', can then be used to change the mental attitude of the agents (their belief, desire or intention). In addition to this, a set of predefined agent interaction protocols is defined, such as the iterated-contract-net protocol, useful for negotiation. It is clear that standardising on an Agent Communication Language alone is not sufficient to achieve interoperability. [33].

4.2.3.2 XML Encoding

By XML the syntactic representation will enhanced (i.e., extend) the (previous) canonical (pure "American Standard Code for Information Interchange" - ASCII) syntactic representation by introducing markup for parsing (the "tags", in XML terminology). This markup significantly facilitates the development effort needed for parsing in and out. The XML representation also facilitates introducing pragmatic/operational elements that go beyond what the pure ASCII previous syntax did: notably, via links (in a similar sense as does HTML compared to ASCII). For example, the ACL message includes information beyond what is equivalent to that. Here, the receiver is not just some symbolic name but is also a URL that points to a particular network location which could provide additional information about the receiver agent's identity (e.g., how to contact its owner, its network ports, etc.).

Encoding ACL messages in XML offers some advantages because the XML-encoding is easier to develop parsers for than any other encoding. The XML markup provides parsing information more directly. One can use the off-the-shelf tools for parsing XML — of which there are several competent, easy-to-use ones already available— instead of writing customized parsers to parse the ACL messages. A change or an enhancement of the ACL syntax does not have to result to a re-writing of the parser. As long as such changes are reflected in the ACL DTD, the XML parser will still be able to handle the XML-encoded ACL message.

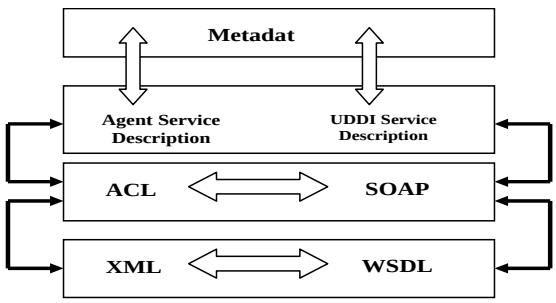

Figure 7: XML-ACL Communication Diagram

In short, a significant advantage is that the process of developing or maintaining a parser is much simplified. More generally, XML makes ACL more "World Wide Web (WWW)-friendly", which facilitates Software Engineering of agents. Agent development ought to take advantage and build on what the WWW has to offer as a software development environment. XML parsing technology is only one example. Using XML will facilitate the practical integration with a variety of Web technologies.

```
<?xml encoding="US-ASCII"?><!ELEMENT
fipa_performative (perfName, sender, receiver,
content, ontology, language)>

<!ELEMENT perfName (#PCDATA)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT receiver (#PCDATA)>
<!ELEMENT content (#PCDATA)>
<!ELEMENT ontology (#PCDATA)>
<!ELEMENT language (#PCDATA
```

Figure 8: Simple ACL-XML DTD

```
  <?xml version="1.0"?>
<fipa_performative>
 <perfName>request</perfName>
 <sender>hisAgent</sender>
 <receiver>myAgent</receiver>
 <content>an_SL0_Performative</content>
 <language>SL0</language>
</fipa_performative>
```

Figure 9: ACL-over-XML Performative

5.1 Scenario Outline

An example of building an agent specialized in searching product on every host of a multi-agent system. The agent has associated a ***ProductTask*** object, which represents the agent scope (objective). This object will receive a reference to a service from the agent's current host. The agent will act like an observer; in case the task fails to run (incompatible service type, protocol error, etc.), the agent will cancel task's execution and will search another service or another host.

To illustrate this idea, we consider virtual shops distributed on the WWW as an example. These sites publish product catalogues, which can be consulted by customers. In the simplest scenario, people use conventional browsers to visit the shops and to order products. Because this is a rather time consuming activity, it is likely that they will limit their exploration to only a few sites. an "Application Program Interface" (API) java abstract class defines product class methods.

In a more advanced scenario, users would not actually visit the sites. Rather, they would interact with a shopping agent and tell him the kind of products they are interested in. The shopping agent would then visit a large number of shops, using an XML-based format to retrieve pricing information. The developer has to override abstract methods to specify particular handling of the desired task. The host will send the request for a service using agent information and task's metadata. Information provided via agent parameter will allow authentication and grant permissions to use this service. The task's metadata will help the services access interface to give the correct result. In this case, the service will use the metadata of the task as a data element with certain fields such as:

Task_type = search, content = product, type = new, metadata = ' product catalogues'

The structure of above code can be also used in other situations, without modifications of the depicted classes; the data element of product catalogue.

In this case the mobile agent is composed of two main parts: execution and support code plus XML metadata. The XML file plays the role of a briefcase that includes information such as agent state, hosts (Internet Protocol (IP) addresses) to visit, and information that the agent has collected. The execution bytcode holds all the intelligence required by the agent to perform tasks and make decisions relating to migration and cloning. The support code comprises the parser that is used to access the XML file and the client handler that allows the agent to attach its files to the SOAP message before migrating. This setup makes the agent more self-sufficient in that it carries with it code that is needed for execution and migration. The agent can be in three modes:

1- Standby mode where it is waiting to be initialized and its files residing in a designated home directory on local host,

2- Migration mode where the code and XML data are bundled into an XML SOAP message as an attachment, and

3- Execution mode where the agent's bytecode is executing on top of the "Java Virtual Machine" (JVM) in its own process.

Whenever an agent migrates to a new host, it adds to its "visited" list the URL of the host, performs its tasks, and then determines whether it should migrate to another host or terminate (based on rules and criteria that are stored in its metadata). To enable the mobile agent to read from the XML file and write information to it, the SAX API (Simple API for XML) provided by the "Java Web Services Developer Pack" (JWSDP) "Java API For XML Processing" (JAXP) API is used to parse the mobile agent XML file. SAX is event-driven and offers a serial access mechanism that does element-by element processing. For example, to look for the next IP address to visit, the agent uses the SAX parser to check the value of the attribute visited under the HostByIP tag, and returns the address of the first host in the list whose visited value is 0.

Once the agent decides to move to a new host, it requests the WSDL file that describes the Web service on the destination host. Having the WSDL file, the mobile


agent creates a dynamic stub accordingly and attaches itself (i.e., all the files constituting the agent) to the SOAP message and then initiates an XML-RPC. Upon receiving the XML-RPC request, the Web service extracts the files from the SOAP message and invokes the class file responsible for the agent execution in a new thread. As it appears, the following tasks are required by the Web service: providing a WSDL file that describes how to invoke it, creating a new thread for each agent, and finally invoking the agent to execute as shown in figure [10].

5.2 Building the Application

The ***ProductCatalog*** class is an implementation of the Catalog interface and provides standard Cataloguing functionality:

```
public class ProductCatalog implements Catalog{
   public static ProductCatalog newInstance();
   public String addNewProduct(String
                        name,float initBalance)
          throws ProductException;
   public Account getProduct(String id);
   public List getProducts();
}
```

This class is the existing class whose functionality we wish to provide access to via a web service interface. Instances of the Catalog interface (e.g. ProductCatalog) are created by a CatalogFactory:

```
public class CatalogFactory {
   public CatalogFactory();
   public Catalog createCatalog();
}
```

5.2.1 Exposing the Catalog as a Web Service Using Metadata Annotations

In order to expose an instance of the existing Catalog as a web service we will:

1. Mark the CatalogServiceImpl class as a Web service using the **@WebService** metadata annotation:

```
@WebService (
   serviceName = "annotatedCatalog",
   targetNamespace =
       "http://service.annotatedCatalog")
public class CatalogServiceImpl {

// service implementation code ...
```

Note that in addition to marking the class as a Web service implementation, the annotation attributes provides the facilities for designating the service name and target name space. These attributes will map to the generated WSDL **<service>** ***name*** attribute and **<definitions>** ***targetNameSpace*** attributes, respectively. Annotate the class methods with **@WebMethod** metadata annotations to expose them as Web service operations; methods that are not marked will not be publicly exposed. The implementation class (CatalogServiceImpl) delegates various calls to a Catalog instance it creates. For example, the createProduct() method simply calls the underlying Catalog's addNewProductt() method (m_Catalog is an instance of Product Catalog obtained from the

2. CatalogFactory). It is these methods that need to be annotated:

```
<portType name="CatalogServiceImpl" >
   <operation name="create-account">
. . .
```

3. Note that in addition to marking the method as a Web service operation, this annotation designates the name of corresponding WSDL operation (***operationName*** attribute). The annotation also provides the ability to specify the SOAP action header with the ***action*** attribute (not shown here). The annotation above yields the following WSDL fragment:

```
@WebMethod (operationName="create-product")
public String createProduct( @WebParam
(name="productName")
String acctName,float initBalance) throws
      RemoteException,ProductException {
   return
 m_Catalog.addNewProduct(prdName,initBalance);
}
```

Annotate the class method parameters with the ***@WebParam*** metadata annotation. This annotation allows the developer to specify the name of the parameter as it appears in the WSDL. For RPC bindings this would yield the name of the WSDL <part> representing the parameter; for document bindings, this is the local name of the XML element representing the parameter (as is the case for this example):

```
@WebMethod (operationName="create-product")
public String
createAccount( @WebParam(name="productName")
String acctName,float initStatus) throws
      RemoteException,ProductException {
   return
m_Catalog.addNewProduct(acctName,initStatus);
}
```

The **@WebParam** annotation also allows the developer to specify whether the attribute is pulled from a SOAP header, the parameter mode (IN, OUT, or INOUT), and the parameter's namespace the annotation


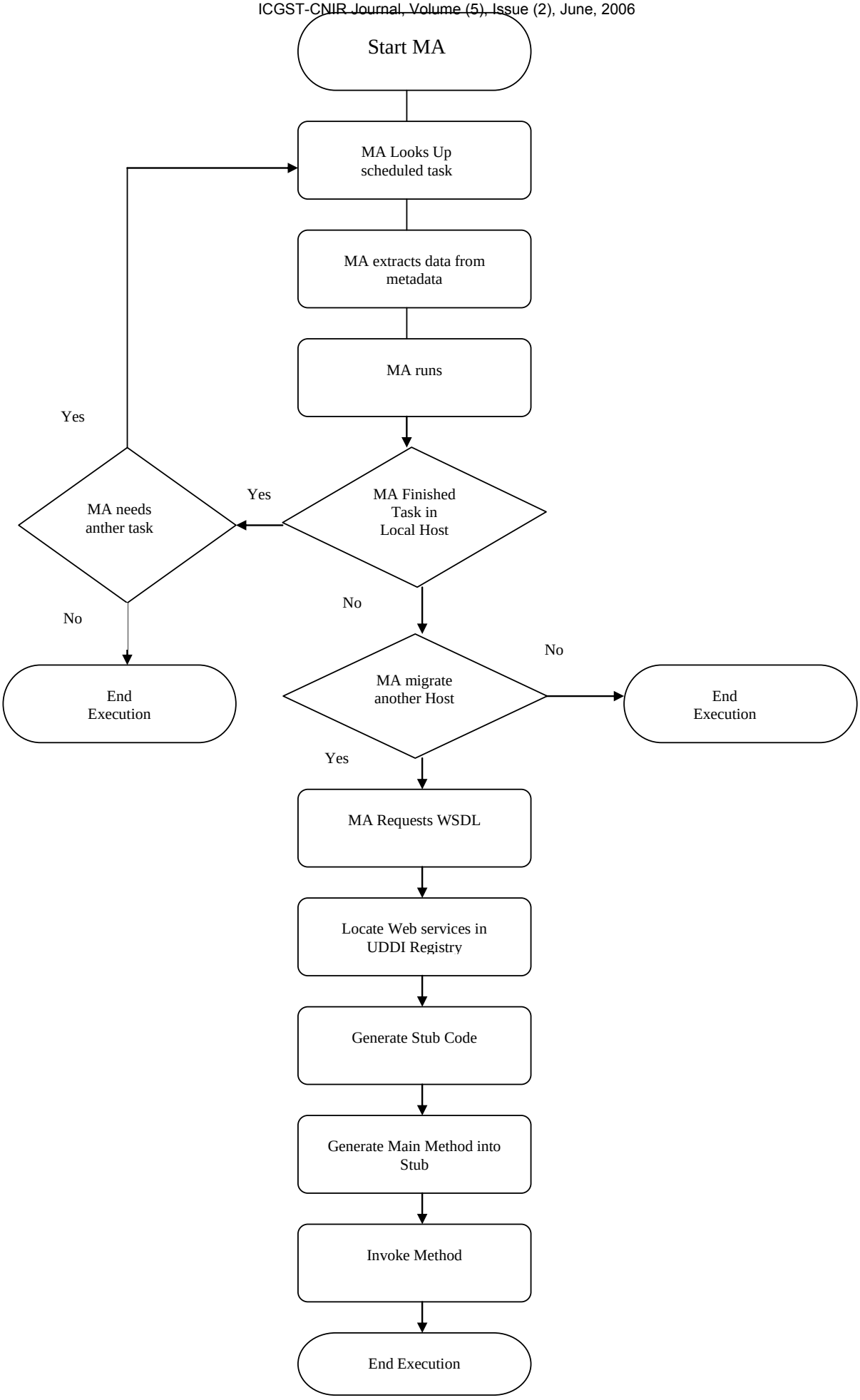

Figure 10. Agent Execution Processing

in this sample yields the following WSDL fragment:

5.2.2 Creating a Client Application

Once the service has been created and deployed a client

```
<complexType name="create-account">
   <sequence>
      <element name="productName" type="string"
nillable="true"/>
      <element name="param1" type="float"/>
   </sequence>
</complexType>
. . .
```

application that leverages the service can be created from the WSDL file generated as part of the service generation

```
void GoodProduct() throws Exception {
      String accountID =
m_endpoint.createProduct(USER1,123);
      // ... print statements removed for clarity ...
      m_endpoint.status(productID,123);
      float status = m_endpoint.getstatus(productID,
USER1);
      System.out.println("Current status is now " +
status);
      System.out.println("Shipinging 5 from product");
      m_endpoint.ship(accountID,123);
      status = m_endpoint.getStatus(productD,
USER1);
   }
```

process. The Oracle Application Server's client generation tool creates, in addition to the classes required by the "Java API For XML - Remote Procedure Calls" (JAX-RPC) runtime [34], a convenience class that shields the developer from some of the more mundane JAX-RPC service instantiation tasks. This class, referred to as a utility client, is leveraged to invoke methods on the remote service by the Cataloging application (CatalogApplication) (m_endpoint is the class attribute for the utility client):

Given the Catalog implementation and support classes and the annotated service implementation we can proceed with the generation and deployment of a web service

6. Conclusion and Future Work

The purpose of this paper is to introduce a theoretical description of the presented framework, which is implementing distributed application "web services" using mobile agents, where, mobile agents are annotated with metadata to describe services, Agent life-time is bound to a service request Communication is restricted to unify inter-agent communication and Web service invocations. Future work will focus on adding Additional services could be developed. Instead of a name service, a global service directory (Registry) could be used, and alpine framework to integrate with semantic Web directions, for example, to provide metadata and ontological support, describing agents, hosts and their interactions in OWL (Web Ontology Language) or OWL-based languages.

Abbreviations:

ACL	… Agent Communication Language
API	… Application Program Interface
ASCII	… American Standard Code for Information Interchange
DDL	… Database Definition Language
DF	… Directory Facilitator
DNS	… Domain Name System
DTD	… Document Type Definition
FIPA	… Foundation for Intelligent Physical Agents
HTML	… Hyper Text Markup Language
HTTP	… Hyper Text Transfer Protocol
IP	… Internet Protocol
J2EE	… Java 2 Platform, Enterprise Edition
JAXP	… Java API For XML Processing
JVM	**… Java Virtual Machine**
JWSDP	… Java Web Services Developer Pack
MASS	… Middleware for Adaptive Semantic Support
OWL	… Ontology Web Language
RDF	… Resource Description Framework
RPC	… Remote Procedure Calls
SAX	… Simple API for XML
SOAP	… Simple Object Access Protocol
UDDI	… Universal Description, Discovery, and Integration
URI	… Uniform Resource Identifier
URN	… Uniform Resource Name
WSDL	… Web Services Description Language
WWW	… World Wide Web
XLL	… Extensible Linking Language
XML	… Extensible Markup Language

7. References

[1] James E. Hunton, Stephanie M.Bryant, Nancy A. Bagranoff ,"Core Concepts of Information Technology," ISBN: 0-471-22293-3 Paperback pp 304 July 2003.

[2] Ahmed Shaaban Abdel Alim, Imane Aly Saroit Ismail, and S.H.Ahmed, "IDSUDA: An Intrusion Detection System Using Distributed Agents," CNIR Journal, pp: 1-11, Volume (5), Issue (1), Dec., 2005.

[3] http://www.recursionsw.com/mobile_agents.htm..

[4] Mike Clark, Peter Fletcher, J. Jeffrey Hanson, Roman Irani, Mark Waterhouse, and Jorgen Thelin, "Web Services Business Strategies and Architectures, "Wrox Press ,August 2002.

[5] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana, "Web Services Description Language (WSDL) 1.1," W3C Note 15 March 2001

[6] John J. Barton, “SOAP Messages with Attachments,” W3C Note 11 Dec. 2000.

[7] James Marshall, “HTTP Made Really Easy A Practical Guide to Writing Clients and Servers,” http://www.jmarshall.com/easy/http/

[8] Kirk A. Evans, Ashwin Kamanna, and Joel Mueller, “XML and ASP.NET,” SAMS, ISBN: 073571200X; Published: Apr 8, 2002;

[9] Tyler Jewell , and David A. Chappell, “UDDI: Universal Description, Discovery, and Integration,” Part 1, O'Reilly March 2002.

[10] Ralph Swick, “Metadata and Resource Description,” http://www.w3.org/Metadata/

[11] Peter J. Bogaards, “Metadata and XML,” EIDC 2004 –Wiesbaden, 10 November 2004 .

[12] Antonio Corradi, Rebecca Montanari, and Alessandra Toninelli , “Adaptive Semantic Support Provisioning in Mobile Internet Environments,” ICDCS Workshops: 283-290 , 2005.

[13] D.G.A. Mobach, B.J. Overeinder, N.J.E. Wijngaards, and F.M.T. Brazier, “Managing Agent Life Cycles in Open Distributed System,” IIDS Group, Department of Artificial Intelligence, Faculty of Sciences, Vrije Universiteit Amsterdam, de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

[14] “Agentspace ,” http://www.cis.strath.ac.uk/~if/agentspace/

[15] Dominic Greenwood and Monique Calisti, “Engineering Web Service - Agent Integration, Systems,” Man and Cybernetics, 2004 IEEE International Conference on Volume 2, pp.:1918 - 1925 vol.2, 10-13 Oct. 2004.

[16] M. Lyell, L. Rosen, M. Casagni-Simkins, and D. Norris. “On software agents and web services: Usage and design concepts and issues,” Proc. of the 1st International Workshop on Web Services and Agent Based Engineering, Sydney, Australia, July 2003.

[17] Craig Thompson, “OMG Agent Working Group Agent Technology,” White Paper and RFP Roadmap, March 14, 2000.

[18] Karen Myers, “Naming Service Specification,” W3C Workshop on Constraints and Capabilities to Explore Next Web Services Layer, October 2004.

[19] Jo Rabin, Charles McCathieNevile,Mobile Web Best Practices 1.0, W3C Working Draft 20 December 2005.

[20] Danny Lange, and Mitsuru Oshima , “Programming and Deploying Java Mobile Agents with Aglets,” Wesley, 1998 (ISBN: 0-201-32582-9) , http://aglets.sourceforge.net/

[21] Graham Moore, “RDF and TopicMaps An Exercise in Convergence,” XML Europe, Berlin. 2001.

[22] Marie Thilliez, Thierry Delot, Sylvain Lecomte, and Nadia Bennani. "Hybrid Peer-To-Peer Model in Proximity Applications," AINA, p. 306, 17 th International Conference on Advanced Information Networking and Applications (AINA'03),IEEE 2003.

[23] Omicini, A.; Zambonelli, F.; Klusch, M.; and Tolksdorf, “Coordination of Internet Agents Models, Technologies, and Applications,” ISBN: 3-540-41613-7, 2001

[24] Paul Albitz, and Cricket Liu, “DNS and Bind,” O'Reilly & Associates, ISBN: 0-596-00158-4, April 2001.

[25] Lee and B. Zheng, “Data Management in Location-Dependent Information Services,” The 20th IEEE Int. Conf. on Data Engineering (ICDE '04), Boston, MA, March 30 - April 2, 2004.

[26] R. Fielding, “Uniform Resource Identifier (URI): Generic Syntax,” Network Working Group, January 2005. http://www.gbiv.com/protocols/uri/rfc/rfc3986.html.

[27] Jakob Nielsen's, “URL as UI,” Alertbox, March 21, 1999, http://www.useit.com/alertbox/990321.html

[28] Yao-Chung Chang, Jiann-Liang Chen, and Wen-Ming Tseng, "A Mobile Commerce Framework Based on Web Services Architecture," *ITCC*, pp. 403-408, International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume I, 2005.

[29] “Knowledge Query and Manipulation Language (KQML),” http://www.cs.umbc.edu/kqml/

[30] “Knowledge Interchange Format (KIF),” http://logic.stanford.edu/kif/kif.html

[31] “FIPA Agent Communication specifications,” http://www.fipa.org/repository/aclspecs.html

[32] Barry Smith, “Towards a History of Speech Act Theory ,” http://ontology.buffalo.edu/smith//articles/speechact.html

[33] Michael Buckland, “Vocabulary as a Central Concept in Library and Information Science,” Third International Conference on Conceptions of Library and Information Science (*CoLIS3)*, Dubrovnik, Croatia, 23-26 May 1999.

[34] Aoyon Chowdhury, and Parag Choudhary, **“**Working with JAX-RPC, Java APIs for XML Kick Start,SAMS,” ISBN:0-672-32434-2