AGORA Framework for Service Discovery and Resource Allocation

Khan, A.Basit
Department of Computer and Information Science (IDI)
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
Email: basit@idi.ntnu.no

Mihhail Matskin
Department of Computer and Information Science (IDI)
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
Email: misha@idi.ntnu.no

Abstract—Integration of Web Services and agent technology is still a problem which needs to be solved. Several different approaches have been proposed and demonstrated, however the proposed solutions mainly targeted to translation of different standards. We believe that the main problem is not translation of standards, but usage of different standards together. In this paper we explore an approach to achieve interoperation of agents and Web services. Instead of performing translation we propose a framework which focuses on using different kind services (including Web Services) and agents together. Our proposed framework supports well known standards and is able to support future changes. By virtue of this work we try to solve three different problems, firstly we provide a solution which unifies different standards, secondly we propose an agent framework which is capable to adopt to changes and evolve in open environment and lastly we adopt a peer-to-peer service discovery and invocation in our framework. Our agent framework is general enough to adopt to different problems and the existing implementation of our framework can be extended/overridden without need to change the core concepts. As a proof of concept we carry out a case study in the area of mobile collaborative learning.

Index Terms—Multi-Agent; P2P; AGora; Services; Service Discovery; FIPA; W3C; Manager Agents;

I. INTRODUCTION

There is a great shift towards Service Oriented model of software design; software components are designed to provide services. These services can be invoked by any other interested party. However, this requires that both the service provider and the service consumer follow the same standards. Several such standards have been proposed by World Wide Web Consortium (W3C) [32]. With increased and open availability of numerous services the volume of information which is produced and which surrounds us is also massive and it is constantly increasing. Furthermore, the process of finding the right service is not a simple task. This situation may be improved by the adaptation of software agents [2], [29], [30], [33] in the software design. The agents may simplify the process of service discover and invocation. However, software agents are standardized by another standardizing body – Foundation for Intelligent Physical Agent (FIPA) [1]. The standards proposed by FIPA and W3C are overlapping. However, FIPA’s agent services description, services advertisement repository and format of communication messages (ACL) [3] are different from that of W3C’s services description (WSDL) [4], services advertisement repository (UDDI) [31] and format of communication messages (SOAP) [20] respectively. This situation poses a major difficulty to integrate software agents and web-services. Furthermore, a central (or federated) service (or resources) discovery repository is a very simplistic solution with the major drawback of having a single point of failure of the whole software system, especially when we consider that the fundamental goals of Service Oriented Architecture (SOA) are to cope with open environments, heterogeneity and dynamism. Above all there exist many different agent frameworks. These frameworks force their own model of system design [5], [13], [14], [26], [27]. The whole situation results in poor integration of software agents and web-services – since the focus of attention is to fuse the web-services technology with agent technology, while there is less emphasis on functionalities provided by the services and internal model of agents and their communication, coordination and negotiation strategies. The resulting software confuses the difference between an agent and agent’s services. The service discovery mechanisms are mainly based on central services repository or hardcoded. The notion of autonomy of an agent and its agency is intertwined with the services (i.e functionality) and vice versa.

To handle the situation discussed above, we present a multi-agent framework called Agent Oriented Resource Management (AGORA). AGORA framework proposes a solution to several problems involving software agents and services. The main objectives of our work are as follows

- Clearer separation between an agent and its services
- Allowing an agent to provide and consume different kind of services; FIPA, RESTful [16], W3C based services
- Emphasis on communication, coordination, negotiation and management aspects of software agent
- A flexible multi-agent system design which can scale in open environments
- Efficient discovery/locating the services in a peer-to-peer (P2P) fashion, without the need of a central repository.
- Promoting adaptation of standard for future extensibility and interoperability

The rest of the paper is organized as follows. First we
provide background information about the existing situation of how interoperation is achieved between Web Services and multi-agent systems; in the light of this discussion AGORA system is positioned. Next section discusses AGORA framework in details. Later we discuss how different AGORAs are connected with each other through parent-child and registration mechanism; here we also elaborate P2P search functionality of AGORA. Next we provide details about AGORA ontology. Implementation details and proof of concept are then discussed, followed by conclusion, limitation and future work.

II. BACKGROUND STUDIES AND POSITION OF AGORA

This section presents a brief overview of related work on software agents and existing solutions to interoperation between W3C and FIPA. We also explain how our work is positioned among existing solutions.

A. Software agents and frameworks

A software agent is an autonomous entity, which represents its user/owner in the digital world and performs certain actions to achieve desired results. Following are the main characteristics of an agent as defined by Russell and Norvig [24]. A software agent always exists in some environment. It can perceive its environment and can act accordingly. Agent communicate with other agent through message passing. KQML [9] and FIPA-ACL [3] are two well known languages which agents can use to communicate. Agents negotiate with each other and this process can be defined as "searching for an agreement" [22]. Software agents plan and coordinate their activities. Each agent has an internal model which can be called as the main logic unit of the agent. The function that maps the input to an agent action is called Agent Function or Agent Behavior (this term is also interchangeably used with Agent Architecture [24]).

Over time many different multi-agent frameworks have been proposed [5], [13], [14], [26], [27]. We do not intend to provide an exhaustive list of all multi-agent framework/architectures which have been proposed. Instead we are trying to highlight the following points

- The main goal of a multi-agent architecture is to provide certain set of services to other agents
- The architecture may or may not follow a metaphor, however, the architecture which follows a metaphor has an advantage over other as the conceptualization of entities becomes explicit
- Most of the architectures enforce their own concepts to be adopted by the agents who want to use services provided by the architectures. However, there is very limited support to extending the architecture in order to provide more functionality. For instance, if the main goal of the architecture is to provide access to heterogeneous resources then it is not possible to adopt the architecture for market based negotiation support.

B. Existing solutions for interoperation between W3C and FIPA

In order to achieve interoperation between FIPA and W3C different approaches have been applied. Almost all of them use a common entity called a Gateway which is responsible for translating the requests/responses originating from both worlds (i.e. world of W3C and world of FIPA) as depicted in figure 1. In this case both the worlds stay isolated from each other and the only bridge between them is the Gateway which can perform translations. Some gateway-based solutions were proposed and shown in [6], [11], [19], [21].

C. Positioning the AGORA system

Our work lies in the intersection of two aspects of software agents discussed above. Firstly we propose a multi-agent framework, which is general enough to be adoptable to a number of situations and domains. Furthermore the framework is extendable and, if needed, it is possible for developers to override the base functionality. This means that the framework provides a high level functionality which can be overridden/extended using "low level" tools (like Java) when needed. Secondly we argue that instead of having a dedicated entity (i.e. GATEWAY) in the system it is worth spending efforts to allow software agents to be able to understand both FIPA and W3C. Not all the standards from W3C need to be understood but only the important ones. Moreover the processing over head involved when GATEWAY will translate all request/responses from all agents will not be much different if all agents were to do it at their own.

III. AGORA FRAMEWORK

AGORA multi-agent framework is based on our previous work [18], [23], [25], [28], [34]. The literal meaning for AGORA (see http://en.wikipedia.org/wiki/Agora) is "place of assembly". People would gather in the AGORA for military duty or to hear statements. Later, the AGORA also served as
a marketplace where merchants kept stalls or shops to sell their goods amid colonnades. Simply said, AGORA serves as a place where different interested parties get together to discuss their interests. The place provides different facilities to coordinate, negotiate and manage the activities. From multi-agent point of view, AGORA follows a metaphor of a marketplace which provides support for conducting collaborative agent activities. Therefore it acts as a space where agents come together to conduct certain activities. The space then provides services such as management, coordination and negotiation related to the activity at hand. Our approach consists of the Multi-agent Architecture and the Engineering methodology for agent based system. However, here we discuss only architectural part of our work with the focus on P2P based resource search. Each AGORA is tailored to support a special kind of functionality, therefore the functional support provided by the sub-components of AGORA is in coherence with the nature of its usability. An AGORA consists of four main components as depicted in the figure 2: AGORA Node, AGORA Managers, AGORA Services and Registered Agents. These components are discussed below.

**Fig. 2. A simple AGORA node**

1) **AGORA Node**: is at the core of everything. This node contains all critical data related to AGORA. This data is shared among all the AGORA Managers. Every AGORA is started with by starting this node. All AGORA Managers and Services are initialized at the start-up of the AGORA node. From the resource point of view, this node can also be considered as a database which contains information about locators of the AGORA Managers (i.e. the contact points where the managers can be reached). It also contains the description and locators of services provided by AGORA. Along with the descriptive information of AGORA (i.e. name, textual description etc), the other information kept in AGORA Node is about Registered AGORAs [see section IV] and Registered Agents [discussed later]

2) **AGORA Services**: each AGORA can provide an arbitrary number of services. These services are different from the services provided by the AGORA Managers (i.e. management related services). The functional context of these services matches the overall context of the AGORA. These services may either follow W3C or FIPA – furthermore an AGORA can provide RESTful services as well. For this reason we generalize the services as resources. From implementation perspective any kind of services can be searched and invoked – any service which implements AGORAService interface becomes an AGORA service. The solution is similar to JAVA collection support. Each service can provide its own description, however in our work we only support WC3, FIPA and RESTful services. Thus, AGORA provides a unifying view of W3C and FIPA. Among the AGORA Managers the Manager Agent is responsible for providing access to services; so it is the host for services. However, Manager Agent is not just responsible for managing services, but it also provides its own services which are treated separately in our architecture.

3) **AGORA Managers**: are agents responsible for providing management support services for an AGORA. All the Managers support FIPA-ACL (with SL [10] as content language) and SOAP messaging.

- **Manager Agent**: is responsible for performing the overall operations related to the management of AGORA. It is responsible for registration and un-registration of other AGORAs and agents (i.e. Registered agents) in the AGORA. It maintains all data in the AGORA Node and insures that the information about the AGORA (i.e services and agent locator) is up to date. Apart from management related activities, this agent provides access to the AGORA Services.

- **Coordinator Agent**: implements logic which insures the smooth flow of collaborative activities. For every activity which AGORA supports there is a workflow describing the activity (i.e. participant A must send a message and in response to that messages participant B must send another message etc). The Coordinator agent insures that all participants of the activity follow the rules prescribed in the workflow

- **Negotiator Agent**: implements logic of conflict resolution for all supported activities of AGORA. By default the contract-net protocol is used. However, other protocols can be provided as well.

4) **Registered Agents**: are external agents who would like to use functionality provided by AGORA. All agents who wish to use the functionality of the AGORA have to be registered at the AGORA. These agents communicate with the AGORA Managers by message passing and consume the AGORA services when needed. Each AGORA can have any number of registered agents. AGORA architecture does not impose any restriction on the internal model of these agent, the only requirement is that they should be able to send/receive ACL or SOAP messages.
The internal model of AGORA Managers is as follows

- Manager: is model-based reflex agent. It has to respond to the incoming requests and manage the AGORA node.
- Coordinator and Negotiator: are goal-based agents. They have an internal knowledge base and their actions are triggered based on the state of environment in their knowledge base. The model of knowledge base is initialized with the ontology which provides possibilities for the agents to reason about their actions.

From programming (i.e., AGORA based development) point of view a base structure of AGORA is provided along with the basic functionality pre-programmed for each component. This implementation includes communication protocols for all the AGORA Managers which are discussed later. The AGORA node services are also part of the implementation. A developer can then extend, add or override the functionality of each AGORA.

IV. AGORA PARENT CHILD RELATIONSHIP TREE

Several different AGORA can be combined together in a network where each AGORA is tailored to provide different services and management functionality (provided by Manager Agents). Upon creation of a new AGORA it is added into a graph of already running AGORAs. This approach allows the system to be extended in an incremental fashion. At the same time any other Agent who may be interested to use some functionality can search and be connected to as many AGORAs as it needs. Since an AGORA can be developed by extending already existing AGORA a programmer can easily create a new AGORA from the existing ones. In this way the complete network of AGORAs can be viewed as a bucket of functionality where any interested party can use or add new functionality.

Whenever the system is started, an AGORA called ROOT AGORA is started in the system as depicted in figure 3. The ROOT AGORA is responsible for common tasks related to management (i.e., GUI updates etc.). Every AGORA has another AGORA as its parent with exception of the ROOT AGORA who is created by the system but not by other AGORAs. This means that an AGORA can only be started by another AGORA and upon start-up the information related to parent and child is exchanged between AGORAs. It is worth mentioning that no references of AGORAs are exchanged – instead the exchanged information is a combination of locators of AGORA Managers, information about AGORA Services and other information such as security keys, description, names etc. All this information is kept in a special data structure called "PublicAGORAInfo." Every AGORA, when starts, publishes/advertises this information (i.e., "PublicAGORAInfo") in the parent-child tree which is managed as a Static structure in the system and is accessible to all other components of the system. Furthermore, at any moment an AGORA can update its profile in the tree as changes occur. The parent-child graph can be searched to discover resources available in the system.

V. AGORA AS P2P SERVICE REPOSITORY

Parent-Child relationship supports a static approach to search of resources. A complementary approach which exists in the system is a peer to peer approach, where each AGORA behaves as if it were a peer node in the system.

Any AGORA in the system can be registered to any other AGORA as required. The main reason for such registration is to consume/provide services. When an AGORA registers to other AGORA it provides information (i.e., "PublicAGORAInfo") about itself. The figure 4 depicts a possible graph of registered AGORAs. In such graph when an AGORA is requested to provide some services, it first looks if it

```
FINDRESOURCE (V)
V ← All Registered AGORAs
Φ ← This AGORAs
V0 ← Old V

1 do if V0 = Φ
2 do skip (Avoid Cycles in graph)*
3 do if Φ has V0
4 then do Return V0
5 else for i from 1 to V.size
6 do j ← V[i]
7 do if j has V
8 then Return j
9 else do Broadcast V to Φ
10 do k ← wait(timeout) getreplies
11 do return k

*Match the request and requester
```

Fig. 5. Simple algorithm for P2P search in AGORA network

of registered AGORAs. In such graph when an AGORA is requested to provide some services, it first looks if it
can fulfill the request. If it can not then it looks into the
cashed information about the other registered AGORAs. If
these AGORAs are also not capable of providing the required
services then the request is broadcasted to all the registered
AGORAs who may have cashed the information about the
provider of the requested resource. This approach is identical
to 2nd generation P2P networks (i.e., Flooding-Based systems)
such as Gnutella. Figure 5 presents the algorithm which is
used to implement such resource discovery strategy. While
the approach looks quite simple, it works pretty well when we
consider that the average number of AGORAs in the system
is not greater than 25.

VI. AGORA ONTOLOGY

Since AGORA considers agent’s communication, coordina-
tion and negotiation in open distributed environments it is im-
portant that the system entities use a standard ontology when
they interact with each other. AGORA system incorporates an
ontology which mainly consists of two parts. Firstly it provides
a formal specification of AGORA’s data model (i.e. what kind
of relationships exists between sub components of AGORA)
as shown in figure 6; inverse of relationships presented in red
is also present in the ontology. Secondly it specifies main
actions which can be performed by agents in AGORA (i.e.
AGORA Managers and Registered Agents). These actions are
preprogrammed in the base implementation of AGORA. The
figure 7 depicts the basic pre-programmed actions which can
be performed by agents in all AGORAs.

![Fig. 6. Basic AGORA ontology](image)

Availability of such ontology allows the AGORA Managers
and the Registered Agents to reason about each other. Fur-
thermore each AGORA Managers’s knowledge base is also
constructed from this ontology. The fragments of this ontology
appear in the messages which are transferred among the
AGORA agent. For example, when an agent wants to register
with AGORA it sends a message requesting Manager Agent
to perform its RegisterAgent action in response to the request.

VII. IMPLEMENTATION FEATURES

AGORA uses JADE (Java Agent Development Environ-
ment) [7], [8] as the middleware layer – JADE is fully FIPA
compliant agent development environment. AGORA Managers
are extended version of simple JADE agents. Manager Agents
of AGORA are different, since the Manager Agent is created
as a simple reflex (hardcoded function) agent. The Negotiator
and Coordinator agents have an internal knowledge base and
reasoning in these agents is supported by a Drools [17] rule
engine. System ontology provides a common data model
for the structure of knowledge base and allows common
understanding of agent communication and actions.

VIII. PROOF OF CONCEPT

This section provides an account of a use-case of AGORA
system for city wide mobile and collaborative learning. We
try to elaborate work architecture for its capability to
search and provide resources on demand. The use case
is elaborated and discussed using our other system called
FABULA (http://www.fabula.idi.ntnu.no/; where we apply
AGORA framework. AGORAs implemented in FABULA are
used as an example here, however, the details about different
AGORAs are not provided and only the names of AGORAs
and short description is given. Interested reader is referred
to our other work [15] for more details. The use case is as
follows.

A learner using a mobile device, is moving around in a city
to learn about the historical places of the city. The learner is
also carrying a mobile device and a user agent is running on
his/her device. This agent appears as a Registered Agent
with an AGORA based mobile leaning system which is available
wirelessly in the city. Upon investigating a historical building
the learner wants to leave a comment about the building,
however, the service which allows to store the comment can
not be invoked since it is not known previously. The user tries
to search for that service – which results in a success and
the comments are stored for the later visitors who might be
interested to know the views of learners who visited it earlier.
The search is performed as discussed in section V.
Figure 8 depicts FABULA’s AGORAs started as separate JADE containers. Each AGORA has its Manager Agents started in the container representing each AGORA. In one of the AGORAs (UserManager AGORA) depicted in the figure an agent appears as a Registered Agent. This agent is not aware of the availability of Comment service. Upon performing the search the user is presented with the service which is provided by the ‘Building Manager AGORA-AGORA’. The figure 9 shows the results.

The immediate next step is to adopt AGORA system to CHORD [12] for efficient search of resources in the AGORA network. At the moment ontological support is a bit limited as the ontology mainly provides a common data model and predicates for message content. The future extensions will focus on reasoning with ontology. Currently only keywords based search of services is possible in AGORA. In the future it will be extended to perform semantic search as well.

IX. ACKNOWLEDGMENTS

This research is funded by Norwegian research council NFR-project (176841/S10). We would also like to thank our project members for their support and cooperation.

REFERENCES


