Maamar, Z., B. Moulin, G. Babin & Y. Bédard, 2000, Agent-based architecture for georeferenced digital library interoperability, Intelligent Automation & Soft Computing,Vol. 6, No. 3, p. 225-232

An Agent-Based Architecture for Georeferenced Digital Library Interoperability

Z. Maamar[1,2,*], B. Moulin[1,2], G. Babin[1], and Y. Bédard[2]

zakaria.maamar@drev.dnd.ca, {moulin,babin}@ift.ulaval.ca, yvan.bedard@scg.ulaval.ca
[1]Computer Science Department and [2]Research Center on Geomatics
Laval University, Ste-Foy, QC G1K 7P4, Canada

Abstract

A Georeferenced Digital Library (GDL) describes several geodocumentary resources available in an organization. The use of a single GDL is rather easy. However, the task becomes much more complex when several GDLs are required to satisfy a user's request. As a solution, we intend to set up Software Agent-Oriented Frameworks that support the interoperability of GDLs by providing users with services that will free them from worrying about information distribution and disparities.

1 Introduction

Georeferenced Digital Libraries (GDLs) are information bases describing several geodocumentary resources (maps, satellite images, etc.) available in an organization. While using only one GDL seems simple, the task becomes very complex when several GDLs are involved [3]. Indeed, each GDL is characterized by its own informational content, its own vocabulary, and its own procedures. In order to help users, we aim at developing an interoperable environment of GDLs in the SIGAL (it stands for "Système d'Information Géographique et Agent Logiciel") project. Such an interoperable environment will provide users with services which will not require any knowledge of the individual characteristics of the interconnected GDLs.

There exists a number of studies in the field of system interoperability, in particular those conducted by the *Object Management Group* (OMG) that aims at adapting object-oriented technology to distributed contexts. In fact, OMG's objective is to set up distributed systems offering services independently from the characteristics of each system. In order to create such systems, the suggested approach is based on *object-oriented frameworks* [5].

When dealing with systems interoperability, two main issues must be considered: *communication mechanisms* and *common knowledge*. The former allows the systems to cooperate efficiently,

*Current Address: Interoperability Group, Information System Technology Section, Defence Research Establishment Valcartier, 2459 Pie-XI Blvd North, Val-Bélair QC, G3J 1X5, Canada.

while the latter allows the resolution of knowledge misunderstanding. In the SIGAL project, we considered the following aspects.

The cooperation of multiple systems implies the sharing of their operations. The Distributed Artificial Intelligence (DAI) field provides principles for an efficient specification of these operations, such as coordination and cooperation. In DAI, researchers have studied a broad range of issues related to the distribution and coordination of knowledge and actions in environments involving multiple entities. These entities, called agents, can take different forms depending on the nature of the environment in which they evolve. A particular type of agents, *Software Agent*, has recently attracted much attention [4]. Software agents are autonomous entities having the abilities to assist users when performing their tasks, to collaborate with each other to jointly solve different problems, and to answer users' needs.

In order to exchange information in spite of system distribution, communication protocols are needed. The results from OMG are oriented towards this type of protocols. In an OMG's perspective, software agents can be thought of as "*smart components*" [5] that can be distributed across networks and that can work together in order to achieve a common goal. In that situation, we can rely on OMG's *Common Object Request Broker Architecture* (CORBA) protocol. OMG refers to these components as *distributed objects* [5]. A distributed object encapsulates *data* and *processing rules* [5]. These rules define the behavior of a distributed object when providing the *services it offers*. CORBA defines how to invoke services of such distributed objects independently from their functional and structural characteristics. In order to associate distributed objects to common situations, authors in [5] define what they call *enchainment suites* for the description of realization operations. These suites can exist not only between objects but also between frameworks. In this context, each framework is viewed as a complex distributed object. When several frameworks interact, it is in fact the objects within these frameworks that interact. From this emerges the need for an *interoperable environment for object-oriented frameworks.*

In order to interact efficiently, interoperable systems should agree on the conventions used for representing and understanding the knowledge they exchange. The formalization of this knowledge is part of the *ontology* domain [1]. To ensure interoperability across systems, we must also resolve potential *knowledge disparities* that might occur. Indeed, each system may be part of an organization. In order to enable these systems to interoperate, we have not only to create a hardware and a communication software infrastructure, but also to harmonize the terminology used in the representation of the exchanged knowledge. The language resulting from an *ontological design* allows an expression to be interpreted unambiguously.

The interoperability process has to keep local systems *autonomous* and *independent*. To reach

this objective, we suggest the introduction of several components, called *software agents* [4]. These agents are the *front-ends* of the systems and have the capability to act on their behalf. Furthermore, these software agents can be of different types. For example, *facilitator agents* identify the systems on communication networks and *task agents* support interactions between the systems. However, given the complexity of managing distributed and heterogeneous environments, we propose to gather these agents into *teams* [3]. Teams of software agents, in turn, evolve within what we call *software agent-oriented frameworks* [3], that expand the classical view of frameworks composed of objects.

In this paper, we present the concepts needed for defining software agent-oriented frameworks and their teams. Each framework is composed of one or several teams of agents which are needed to provide services to users. To illustrate these software agent-oriented frameworks, we will describe their use in the particular case of the SIGAL environment.

The paper is organized as follows. Section 1 proposes an overview of our theoretical and practical study and also, introduces the relevant aspects of the interoperability problem and the associated research domains. Section 2 presents the basic notions of software agent-oriented frameworks. Sections 3 to 6 describe the SIGAL environment. Section 7 summarizes the main results brought out by our study.

2 A Software Agent-Oriented Framework

In this section, we briefly present the main characteristics of a software agent-oriented framework and its operating mode [3].

A *software agent-oriented framework* offers a set of *services* (they are operations to perform in order to satisfy specific needs) that can be requested either by users or by other frameworks. A framework is an environment composed of a *supervisor* and one or several *teams of software agents*. The services provided by a framework are performed by different teams set up by the framework. These teams are composed of agents selected from a *bank of software agents*. These agents have different functionalities which are specific to the application to be developed.

Teams of agents are structured differently according to their responsibilities in the framework. A team is characterized by a *supervisor* and a set of roles that agents must fulfill according to their capabilities. A role is identified by a list of parameters: *role label* (such as coordination, resolution), *results to provide*, and offered and required *services* when playing the role. Considering these parameters, a framework selects appropriate agents from the bank of software agents. To carry out their operations, interactions may involve agents from different origins: (same team,same frame-

work), (other team,same framework), or (other team,other framework).

Services provided by a framework satisfy specific users' needs such as information retrieval. When a service is invoked by a user, the framework's supervisor agent activates a *realization scenario*, which specifies the functional and structural characteristics of the teams of agents that will perform various operations required to carry out the service. According to that realization scenario, the framework supervisor creates teams that will play roles specified in the scenario. At their own level, team supervisor agents activate realization scenarios in order to coordinate the activities of their software agents.

3 The SIGAL Project

An application of software agent-oriented frameworks is the SIGAL project, in which we are developing an interoperable environment for GDLs.

Within an organization that manages spatially referenced data, several types of documents are used to describe the presence and the nature of these data. Managing such documents is a complex task. Hence, GDLs can be very useful and helpful. GDLs are information bases describing the available geodocumentary resources. As a result, GDLs can improve knowledge of the nature of data and identify the responsibility of "who does what, when, and how".

A survey, conducted in July 1996 [7], sheds light on the heterogeneity of current GDLs. If a person visits these GDLs, she will easily notice that: the content of the GDLs differs from one to another; the data standards are quite different, or even absent in some cases; different words are used to represent the same concept and *vice versa*; the user interfaces always differ; etc. All these differentiating elements would need that users adapt to each GDL's requirements and understand their different information and structural characteristics; which is an impossible task. The SIGAL environment aims at solving these problems.

In the following sub-sections, we describe the SIGAL environment in terms of architecture, agents, and types of frameworks.

3.1 SIGAL's Architecture

The SIGAL environment (Fig. 1) is an interoperable architecture characterized by the use of a *client/server* approach, the implementation of *mirror sites*, the use of three types of frameworks (server, client, local-source), and the dynamic generation of client frameworks, based on users' needs [3].

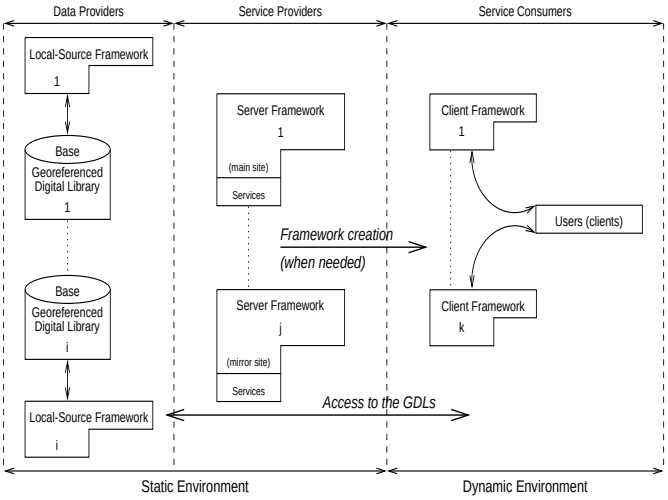

Figure 1: Multiframework architecture of the SIGAL environment

The local-source frameworks maintain the autonomy and the independence of the GDLs. Therefore, they interface GDLs with communication networks and process the data requests sent by client frameworks.

The server framework is the backbone of the SIGAL environment; it monitors all the operations needed to support services offered to users and to other frameworks. To avoid *overloading* the server framework, we suggest to duplicate it on *mirror sites* and consequently, to define *reliable update protocols*.

When users need information from several GDLs, they invoke relevant services from the server framework. This invocation initiates the creation of a *client framework* on the *user's machine*. Next, the server framework delegates operations to the client framework and limits its involvement to the monitoring of these operations.

3.2 SIGAL's Agents

In order to meet the SIGAL's architectural requirements (Section 3.1) and to perform the various services offered by SIGAL (Section 4), we have identified different types of agents [3].

Supervisor-Agent - it monitors the performance of operations needed to fulfill the various services at a local level (within a framework) and at a global level (between frameworks).

Domain-Agent - it resolves knowledge disparities by using the common knowledge of the Ontology Base.

Help-Agent - it presents to users the services which are offered by server frameworks.

Interaction-Agent - it mediates interactions between either teams or frameworks, by handling the

exchange of informational and structural components required by the frameworks and their agents.
Interface-Agent - it assists users in formulating their needs using the knowledge and vocabulary of the Ontology Base.
Knowledge-Agent - it knows the protocols through which a GDL accepts requests and provides informations back. This agent also monitors changes occurring in the GDL during local knowledge updates.
Resolution-Agent - it processes user's requests. Therefore, the resolution process may require a decomposition of the request into sub-requests which are sent to the relevant GDLs.
Scenario-Agent - it manages the realization scenarios that specify the procedures needed to carry out the services offered by the SIGAL environment.

3.3 SIGAL's frameworks

Three types of frameworks are the basis of the SIGAL environment.

The Server Framework

The *server framework* is responsible for carrying out services required by users, or necessary to maintain SIGAL's architecture. The server framework contains a single team composed of several agents: *Supervisor*, *Help*, *Scenario*, *Interaction*, and *Domain*. Moreover, this team has access to three informational resources; the *Service*, *Scenario*, and *Ontology* Bases.

The Service Base is accessible through the Help-Agent and contains a list of user and SIGAL services offered by the server framework. The Scenario Base is accessible through the Scenario-Agent and contains the realization scenarios available to the team. The Scenario Base contains several components: (1) formulation patterns used to help a user specify his needs, (2) characteristics of agent roles needed to carry out the scenarios, and (3) procedures used to build the plans that are executed by the agents during the scenario. The Ontology Base provides a detailed description of the knowledge contained in each GDL.

The Client Framework

The *client framework* is created by a server framework, when accessed by a user requiring a service. The client framework is set up *temporarily* on a user's computer.

The client framework is composed of a *Supervisor-Agent* and of one or several teams according to the number of GDLs necessary to process user's request. The Supervisor-Agent is created by the server framework. In turn, this agent creates the *main team* of software agents and assigns it

the activities prescribed in the scenario which identifies the invoked service. When required, the Supervisor-Agent also creates secondary teams.

The main team is composed of three agents (Interface, Resolution and Interaction) and is responsible for processing the user's request as well as for identifying which GDLs must be accessed. Based on the number of GDLs needed, the main team asks the Scenario-Agent of the server framework for additional agents. One or several *secondary teams* are created; each containing an Interaction-Agent and a Resolution-Agent. The main team is the only team allowed to communicate with the server framework. Each team (main or secondary) interacts with a specific local-source framework in order to solve the sub-problem at hand.

The Local-Source Framework

Each *Local-source framework* interfaces with a GDL of the SIGAL environment. It is assigned to a specific server framework in order to keep it informed about the changes occurring in the informational content of the GDL. When such a change occurs, the Domain-Agent of the server framework updates its *Ontology Base*; these updates are then propagated to the other mirror servers and are specified by *update scenarios*, that belong to SIGAL services. A local-source framework also processes *data requests* directed to its associated GDL.

The local-source framework is made up of three agents set in one team of agents (Supervisor, Interaction, and Knowledge). The *Supervisor-Agent* sets the other agents and monitors the operations performed in the local-source framework. The *Knowledge-Agent* interacts with the GDL in order to get the data requested by the *Resolution-Agents* of the client framework. This data is transmitted by the *Interaction-Agent*, which allows the local-source framework to interact with server and client frameworks.

4 Services Offered by the SIGAL Environment

Two categories of services are offered by server frameworks: *user* and *SIGAL*.

User Services

A *user service* fulfills a need that involves several GDLs. The user formulates his request independently from the distribution and heterogeneity constraints of the GDLs.

The user consults the list of services proposed by the Help-Agent. When a service is identified, this agent initiates its realization by informing the Supervisor-Agent of the server framework.

The Supervisor-Agent performs the operations characterizing the service and asks the Scenario-Agent to select the relevant scenario. After the scenario has been identified, the client-framework components are selected and sent to the client's platform by the Interaction-Agent. The client framework is composed of several agents (Supervisor, Interface, Resolution, and Interaction) and must fulfill user's request. Once this request has been formulated, the Interface-Agent sends it to the Resolution-Agent, in order to determine which GDLs are needed to satisfy the request. To perform this operation, the Resolution-Agent uses the information provided by the Domain-Agent of the server framework. Depending on how the selected GDLs are distributed, the Resolution-Agent asks the Scenario-Agent for new teams for the benefit of the client framework. The number of agent teams set up in the client framework depends directly on the number of GDLs to be requested; in fact, each team (including the main team) is assigned to only one local-source framework. The goal is to request relevant data from these selected GDLs. After processing these data, all the teams send their results to the Resolution-Agent of the main team. The final answer is sent to the user by the Interface-Agent services.

SIGAL Services

SIGAL services support operations that allow insertion or deletion of a GDL, as well as the modification of GDLs' informational content.

Modification of GDL-Informational Content

Any change occurring in a GDL must be pointed out by its local-source framework to the assigned server-framework. The objective is to keep a total coherence between the informational content of the GDLs and that of the Ontology Base. The Knowledge-Agent of the local-source framework monitors the behavior of a specific GDL. During each update, it contacts the Interaction-Agent of the associated server framework. Once it gets this information, the Interaction-Agent sends it to the Domain-Agent of the server framework. Subsequently, this agent performs the maintenance operations and transmits a request for spreading the update to the other server frameworks thanks to the Interaction-Agent. Since the SIGAL environment brings about a total replication situation, it is important to keep the Ontology Base coherent by managing confirmations of updates.

New GDLs Insertion

Any new GDL that has to be integrated into the SIGAL environment is identified by a local-source framework which is assigned to a server framework. The insertion of a new GDL is triggered by the invocation of the appropriate service of the server framework, which is done by the administrator responsible for managing this framework. According to this service, the Help-Agent has to contact the Supervisor-Agent to inform it about the need for setting a scenario modifying the SIGAL's

architecture. This scenario is managed by the Scenario-Agent and is available in the Scenario Base. The realization scenario in this context is characterized by two elements: processing programs and formulation patterns for the Domain-Agent of the server framework, and processing programs and software agents for the new local-source framework. After sending the components of the new local-source framework and after its assignation to a server framework, the Scenario-Agent sends the update scenario to the Domain-Agent in order to update the Ontology Base. When these update operations have been executed, the Domain-Agent sends a request for spreading the update to the other server frameworks thanks to the Interaction-Agent.

5 The Ontology of the SIGAL Environment

When defining the domain ontology, we aim at solving knowledge disparities that may occur when heterogeneous systems interact. By establishing an ontology, we offer a common terminological basis for the various interconnected systems, hence reducing the risks that users get inconsistent information.

In SIGAL's case, ontological disparities exist at different levels. First, being generally developed independently, GDLs may use different terms to describe their data: these different ontologies make it difficult for users to consult several GDLs simultaneously. Second, current GDLs do not help users when formulating their requests. Moreover, a user has to express his needs according to his own vocabulary and understanding of the application domain. Unfortunately, a user is not always able to get the information he is looking for, because GDLs cannot adapt to these terminological differences. In SIGAL, the frameworks cooperate to solve the problems resulting from *ontological disparities*.

When defining the SIGAL ontology, we have used the concept of *meta-data* [2], defined as *data which describe data of the analyzed domain*. The description allows the specification of the data structures, their domain values and their functional and semantic interpretations. We have developed a meta-data model which provides a concise description of the information manipulated by each GDL. This model is called Ontology Base and is managed by the Domain-Agent of the server framework.

6 The Prototype of the SIGAL Environment

Our objective is to set up an *interoperable environment* for GDLs. In order to reach this objective, we have developed software agent-oriented frameworks and spread them across a network. To

manage the evolution of these frameworks, we need communication protocols that enable us to specify messages and data exchanges.

In the development of the SIGAL prototype, we used the Object Request Broker (ORB) *VisiBroker for JAVA* developed by Visigenic Company, JAVA as a programming language (particularly, its applet mechanisms), and the Java Database Connectivity (JDBC) driver [6] in order to connect our several informational resources. We implemented the three types of frameworks which compose a distributed system. Currently, the SIGAL prototype is dedicated to the realization of user services. The following are relevant aspects of this prototype:

1. The client framework is a Java applet that runs on the user's computer. This applet, which integrates the JDBC driver, is able to invoke the server framework services and to send users' requests to the local-source frameworks after their specifications are obtained from the server framework.

2. The server framework is developed on a Web site. This framework uses the JDBC driver in order to connect the Ontology Base.

3. The ORB VisiBroker for JAVA is used to establish communications between the client applets, the server frameworks, and the local-source frameworks.

4. Microsoft's Access database system is used to manage the Ontology Base.

5. All the agents are implemented as Java classes.

7 Conclusion

In this paper, we presented the major characteristics of the SIGAL project which aims at making several GDLs interoperate. The results presented are part of a larger project whose objective is to propose a design method for interoperable environments based on software agent-oriented frameworks [3].

A framework is made up of teams of software agents that are able to fulfill services offered to users. These teams can be set up in a unique framework or in several frameworks leading in the latter case to the creation of an interoperable multiframework environment.

The SIGAL project has been the object of continuing research. We are currently testing the framework architecture described in this paper. SIGAL's prototype uses JAVA as a programming language, the JDBC driver to connect the available informational resources, and finally the ORB VisiBroker for JAVA to specify the behavior of the SIGAL environment's components during distributed operations.

References

[1] T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Technical report, Knowledge Systems Laboratory, KSL 93 – 04, Stanford University, Stanford, California 94305, August 23, 1993.

[2] C. Hsu. *Enterprise Integration and Modeling — the Metadatabase Approach.* Kluwer Academic Publisher, Boston, Mass., USA, 1996.

[3] Z. Maamar. *Contribution à la résolution des problèmes d'interopérabilité des systèmes, une méthode de conception par frameworks orientés-agents logiciels.* PhD thesis, Computer Science Department, Laval University. Québec, Canada, October 1997.

[4] H.-S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(3):1–40, September 1996.

[5] R. Orfali, D. Harkey, and J. Edwards. *The Essential Distributed Objects Survival Guide.* John Wiley & Sons, 1996.

[6] P. Patel and K. Moss. *Java Database Programming with JDBC.* Coriolos Group Books, 1996.

[7] M.-J. Proulx, Y. Bédard, F. Létourneau, and C. Martel. Catalogage des données spatiales sur le world wide web: Concepts, analyse des sites et présentation d'un géorépertoire personnalisable georep. *Revue Internationale de Géomatique*, 7(1), 1997.