University "Alexandru Ioan Cuza" of Iaşi
Faculty of Computer Science

TECHNICAL REPORT

ADF – Abstract Framework for Developing Mobile Agents

Ovidiu Nichifor **Sabin Buraga**

TR 04–01, August 2004

ISSN 1224-9327

Universitatea "Alexandru Ioan Cuza" Iaşi
Facultatea de Informatică
Str. Berthelot 16, 6600-Iaşi, Romania
Tel. +40-0232-201090, email: bibl@infoiasi.ro

ADF – Abstract Framework for Developing Mobile Agents

Ovidiu Nichifor Sabin Buraga

Faculty of Computer Science,
"A. I. Cuza" University of Iaşi
Berthelot, 16 – Iaşi, Romania
{diaboliq,busaco}@infoiasi.ro

Abstract

In this technical report, we propose an abstract framework – *ADF* (*Agent Developing Framework*) – to be used in designing and implementation stages of building (mobile) agents within a multi-agent architecture. We also investigate the possibility of developing, in a generic manner, different components to be integrated into agent applications. The ADF framework consists of a hierarchy of classes (structured as a Java package) and a visual developing tool. The proposed API offers the core functionalities of an agent-oriented system (e.g., agents, hosts, name service, migration, and messaging).

1 Introduction

The Internet became a distributed environment for sharing information, without concerns for its type. Because of its spreading, the information can hardly be found and processed, in an uniform way, by heterogeneous applications.

Agent paradigm – especially, mobile agent paradigm – is one of the promising technologies for information retrieval in general and for certain types of resource discovery in particular. Using agent-oriented distributed systems we can divide the problem into smaller parts (objectives), which can be solved independently, thus achieving to problem modularization and robust distributed calculation. Mobile agents, briefly presented in section 2, are programs that have the ability to migrate autonomously among different interconnected computers (hosts).

In section 3 of this technical report, we describe an abstract framework – *ADF* (*Agent Developing Framework*) – which gives support for building application-transparent and platform-independent components of a multi-agent environment, including mobile agents, hosts and their services. The defined API (Application Programming Interface) consists of a hierarchy of abstract classes, providing a simple programming model and using current standards. The section 3.5 presents certain results of using ADF system in a concrete context.

In the last section, we present the overall picture of the framework, including general conclusions, and we sketch some possible directions of future research work in this area.

2 Background

Agents [6, 7, 11] are software engineering unit, typically larger than a class or module, which can encapsulate roles and goals, and which can be composed in order to build other applications. Agents are able of complex interactions (including negotiation and collaboration), then forming a multi-agent system. These interactions allow agents to adapt their behavior to the prevailing environments. From a communicative point of view, agents are entities that communicate using an agent communication language structured around a set of message types (preformatives), message content schemas (ontologies) and message meta-information.

An important step towards *Internet/Web Computing* is represented by

the mobile computations. A mobile object, usually called a *mobile agent*[1] when operating on behalf of a user, is a downloadable, executable object that can independently move (code and state) at its will – the mobile agent is not bound to the system in which it began the code execution and can travel from one node on a network to another.

Mobile agents present the following important attributes [6]:

- *reactive* (the ability to respond to changes within agent environment),
- *autonomous* (the mobile agent is able to exercise control over its own actions),
- *goal-oriented* (the agents have a planned itinerary, they do not simply act in response to the environment),
- *communicative* (the ability to communicate with other agents, by exchanging information/knowledge),
- *mobile* (the mobile agents can transport themselves from one host to another).

Mobile agents provide a way to think about solving software problems in a networked environment that fits more naturally with the real world. Mobile agents can be used to access and manage information that is distributed over large areas [6, 11].

The main benefit is that the software components can be integrated into a coherent and consistent software system – e.g., a multi-agent system – in which they work together to better meet the needs of the entire application (utilizing autonomy, responsiveness, pro-activeness and social ability).

A mobile agent architecture provides the "framework within which mobile agents can move across distributed environments, integrate with local resources and other mobile agents, and communicate the results of their activities back to the user. This framework can then be used to build mobile agents that perform user-driven tasks to fulfill distributed information management goals" [11].

Within an agent framework, the communication can be established by an agent in order to exchange information with other agents or hosts (e.g., to find agents, to have access to resources, etc.) or by an host in order to send/receive information to/from other hosts (for example, in the case

[1] According to [15], the term "mobile agent" was introduced by Telescript system, which supported mobility at the programming level.

of routing messages between agents that are stored on different hosts). Of course, the host environment might want to establish communications to the mobile agents [4].

Taking this notion further, mobile agents could be used to monitor the network activities and provide input to QoS (Quality of Service) and global optimization mechanisms. They could be used during negotiation (with representative agents) to solve different constraint optimization problems.

The main characteristics of multi-agent systems are [6]:

- each (mobile) agent has incomplete information, or capabilities for solving the entire problem, thus each agent has a limited point of view regarding the overall context;
- there is no global control;
- data is decentralized;
- computation is asynchronous.

The current mobile multi-agent systems [14, 15] – available as commercial or open-source applications – are implemented in various programming languages, such as C++, Java, Tcl/Tk, Scheme or Python.

One of the noticed difficulties – addressed in this paper – is the existing of multiple and incompatible APIs provided by such as systems. Another problems are given by the use of proprietary protocols and the lack of support for another related technologies (e.g., semantic Web or Grid).

3 ADF (Agent Developing Framework)

3.1 Goals

This section explains the overall goals of the new proposed software agent framework. The first step in creating an agent-based application is to define the services that will be provided by host servers to visiting agents. The agent server needs to verify an agent's identity, to create an execution environment, to grant access to its local resources, plus to allow easy agent migration.

3.1.1 Goal Description

The framework must achieve the following [4, 21]:

1. *The framework should provide a simple programming model and a simple API to create new agents*

   Our main goal is to create a simple, scalable and easy to use API. Developers should not have to use large and possibly complex APIs in order to create agents. In addition, they should not be restrained to a specific way of programming, unless they have to consider other aspects, such as security or scalability.

2. *The framework should provide support for messaging*

   The framework should provide both synchronous and asynchronous messaging. Asynchronous messaging is particularly interesting from the point of view of scalability (especially in the context of multi-agent systems), and synchronous messaging is interesting from the developer's point of view. The designed system should support the basic primitives for messaging (and routing, too), enabling specific abstractions and enhancements based on those primitives.

3. *The framework should provide support for agent migration*

   Agents can be static or mobile. Static agents execute on a single host while mobile agents can migrate to other hosts on the network. Static agents are a special case of mobile agents that stay within the host where they are created. To support both static and mobile agents, the framework should provide the ability to give agents the possibility to migrate to arbitrary hosts.

4. *The framework should provide application scalability and adjustment*

   The proposed framework should be as extensible as possible and should also (alike) provide application extension support.

5. *The framework should provide a secure execution environment*

   The security issues [4] are very important, for agents and for hosts, too.

6. *The framework should be built using current standards*

   This aspect will guarantee maximum interoperability between hosts and the portability of the existing agents as well as a new range of further implementations.

3.2 Purposes of ADF Architecture versus Related Platforms

In order to provide a platform-independent software agent platform, the implementation is using Java language. Also, our efforts are focused to create a simple and extensible API, ensuring scalability and flexibility of the developed applications. By providing Java packages that encapsulate the (abstract) API, the framework can be considered a reusable and extensible architecture.

The host architecture follows the model used by Aglets [12] framework, organizing agents in different contexts, to assure a good delimitation of their activities and a better hierarchy.

Like other existing approaches, the naming conventions for the objects (agents and servers) are based on the URI (Uniform Resource Identifiers), in order to ensure the location independence of the accessed objects.

To find agents, a naming service abstract model is provided. The model we choose is inspired by Ajanta [17]. We use a global naming service which also incorporates a global messaging service. This approach can be considered an easy method to keep tracking of agents and their communication, in contrast with the Aglets model [12] (it supports remote communication between agents, but the hosts are not able to redirect the messages to their new locations, if the agents move across the network).

The ADF framework supports messaging and migration facilities. The messages are used in agent-to-agent communication, similar to other existing systems. The service of agent-to-host and host-to-host communication is implemented by using events. Another solution is to use RMI (Remote Method Invocation) – for example, Voyager [24]. The further versions will considered RMI technique, because is more efficient than the one that involves events.

The current implementation is using available standards, like other related systems (e.g., Aglets [12], Ajanta [17], Omega [2] or Voyager [24]). The communication between agents uses XML[2] family, following our previous research in this area (for example, consult [9]). We also use an original data-serialization based on SOAP (Simple Object Access Protocol) – see [3] for details.

From the architectural point of view, our proposed API has some similarities with Tryllian's AFC (Agent Foundation Classes) [23], that provides libraries to allow Java programmers to design and build agent-oriented com-

[2]Consult [25] for the actual standards regarding XML (Extensible Markup Language) technology (syntax, validation, namespaces, transformation, etc.).

ponents. Like our proposal, Tryllian provides a graphical agent-building tool, called VAD (Visual Agent Designer). All Tryllian's components are only available as commercial products.

Another similar project is FIPA-OS (The Foundation of Intelligent Physical Agents – Open Source) [22], a Java-based component-oriented toolkit for building FIPA [21] compliant agents, using different available components.

The existing ADF platform provides a minimal security model for servers. Because the API is extensible, a strong security mechanism (including sophisticated authentication and cryptographic methods) can be added.

We conclude with the idea that a successful agent framework must use the existing standards, must offer a simple and flexible integration with other similar systems and must provide a simple agent programming model. Another key feature is to offer support for complex security services.

3.3 Platform Design

3.3.1 Design Rationale

The aim of ADF project is the design should support diverse implementations. The framework must not be viewed as a complete implementation, but mainly a design of interfaces and contracts. Next versions will complete the demands which were not satisfied entirely in this version. Thus, the framework defines what agent developers need to know in order to make their agents run on any host. It also specifies what a host provider requires in order to assure the execution of the agents within their environment.

A host can be very simple or very advanced and it can provide anything from a few simple services to a rich amount of complex services [21]. Systems can be enhanced with more powerful host implementations due to changing requirements, such as increasing fault tolerance and scalability needs.

The only aspect that developers are required to know is how to make their agents interact with the surrounding environment (agent context), what contracts their agents are required to uphold, and how the agents can acquire the services that they need. The internal logic of the agents (e.g., computational or life-cycle models) is entirely left to be implemented by the programmer.

3.3.2 ADF in the Context of Agent-oriented Software Engineering

Actually, in the context of agent-oriented software engineering, specific areas of interest have included [13]:

- requirements engineering for agent systems;

- techniques for specification of (conceptual) designs of agent systems;
- verification techniques;
- agent-oriented analysis and design;
- specific ontologies for agent requirements, agent models and organization models;
- libraries of generic models of agents and agent components;
- agent design patterns;
- validation and testing techniques;
- tools to support the agent system development process (e.g., agent platforms).

Some of the mentioned areas are covered by our proposed framework. From this point of view, ADF can be considered as an useful experiment of designing agent applications, following [18]. All components of the system and their relationships were designed via UML (Unified Modeling Language) [5] diagrams.

To build a software architecture, the principle of abstraction must be applied: hiding some of the system's details through encapsulation in order to better identify and sustain its properties. Similarly, during the design process of the ADF system, we tried to hide some of the details in order to focus on most important components. These components are described in the next section. Because the system is complex, the provided API – detailed in section 3.4.5 – offers different levels of abstraction. The multiple operational phases are depending on the agent-oriented application to be developed and must be implemented by the programmer.

3.4 Implementation

3.4.1 Framework Overview

According to section 3.1, we are now presenting what requirements are met, explaining how we can resolve these requirements.

The fundamental blocks of an agent system are *agents*, *hosts* and *services* [4, 21]. An overview of the system is presented in figure 1.

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

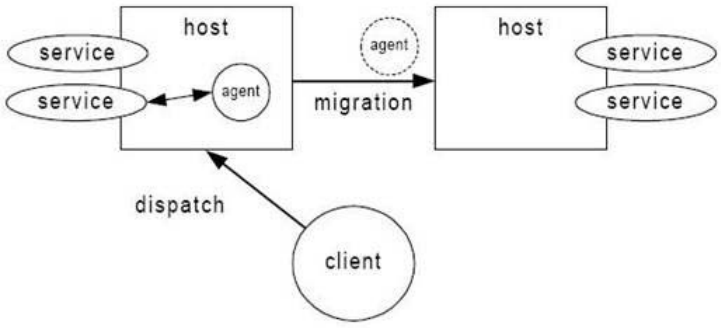

Figure 1: Components of the agent system

services return arbitrary information that can be processed by agents. The proposed framework does not impose any restriction regarding the implementation of needed services.

The two main parts of an agent framework are, of course, 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) [4, 6]. 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 [21].

Another important component of the framework is the *naming service*. 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. Naming generally involves assigning a location independent name to each agent.

This facility is based on the *naming convention* and the *naming service* [4]. Naming convention specifies what an agent's name is, and how it is used (like Aglets [12], our approach uses 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

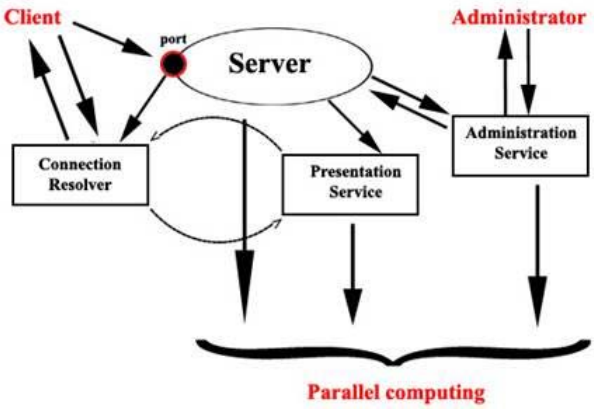

Figure 2: Server concept in ADF

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 ADF 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 – for example, we can develop BDI (Belief-Desire-Intention) [6] architectures.

Next sections will describe the most important ADF elements.

3.4.2 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. 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 RDF constructs [9]). 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.) [8].

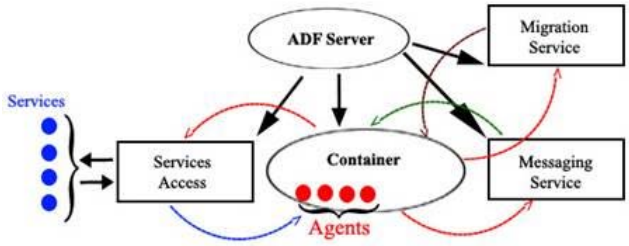

Figure 3: Host concept in ADF

3.4.3 Host

The host is also encapsulated by an abstract class, based on a server abstraction (see figure 2). 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.

The structure of agent host uses different basic concepts, such as:

- *Containment* (a container for the hosted agents),
- *Service access* (an interface to access the host services),
- *Messaging support* (provides communication between agents/hosts),
- *Migration support* (feature to support mobile agents).

More information about these services can be found in [4].

These components were implemented in abstract classes, which will run in parallel within the host context (see figure 3). Each such as abstract class is denoted by a Java service using events and listeners to communicate and launching exceptions to signal special (important) conditions.

3.4.4 Naming Service

We are proposing an abstract scheme that should represent a service name. We expect that any service name should be composed by these two components: *Connection service* (an abstract connection to a database that will be used to store agent locations) and *Messaging service* (an abstract service that should provide messaging support for agent/host communication; ADF offers synchronous and asynchronous messaging). The naming service is depicted in figure 4.

Of course, additional services could be developed. Instead of a name service, a global service directory (registry) could be used.

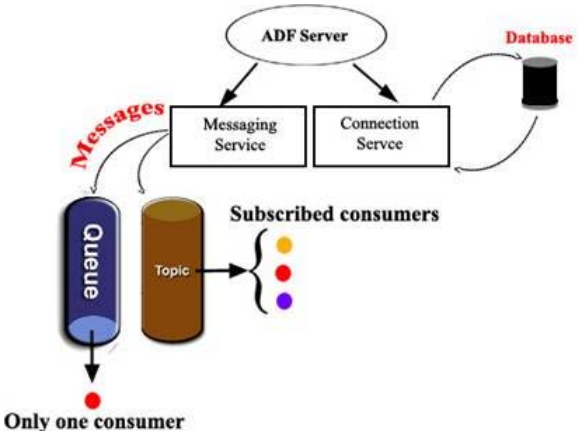

Figure 4: Name service concept in ADF

3.4.5 Implementation Details

The agent framework was built using Java platform. We choose Java because it provides a good class hierarchy, allowing classes to be encapsulated in packages which offer a good representation of abstract schemas. One of the important feature of Java is the platform-independence. Object serialization could be used in agent migration process, thus the state of the agent could be serialized and dispatched between hosts. Java also provides an excellent support for XML (Extensible Markup Language) processing. The XML family could be used in configuration, in communication (messages from/to agents/hosts) and in agent metadata (for details, see [3] and [9]). We use multi-threading, network programming and user-interface facilities provided by Java, too.

The actual implementation of the ADF system is based on JRE (Java 2 Runtime Environment) 1.4.2, JDK (Java 2 Software Development Kit) 1.4.1 and J2EE (Java 2 SDK Enterprise Edition) 1.3.1 [20].

The API provided by the ADF framework is organized in a package named *adf.jar*. Its structure is illustrated by table 1.

3.4.6 ADF Overall Architecture

In fact, the ADF framework is structured in layers. The lowest layer is represented by the JVM (Java Virtual Machine) running on a specific platform (operating system) and the highest layer is populated by different agent-oriented application built in ADF.

The *ADF Host* layer can be viewed as an agent runtime environment.

`adf.*`	ADF package
`util.*`	different useful classes
`server.*`	server-side interfaces
`Host.*`	agent host
`NS.*`	name service
`client.*`	client-side interfaces
`Agent.*`	agent logic

Table 1: Structure of the *adf.jar* package

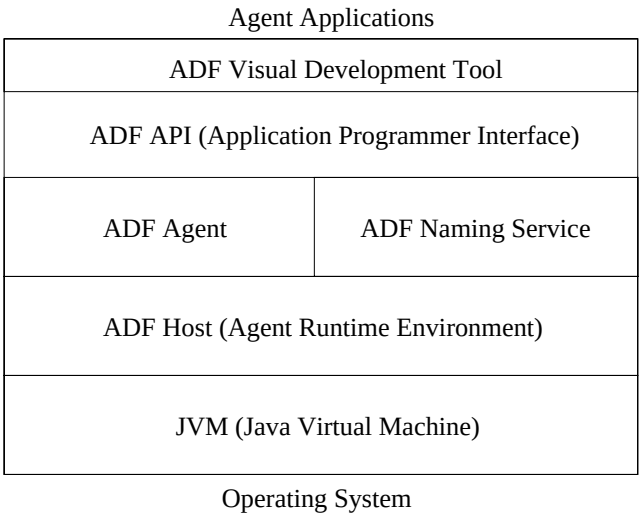

Figure 5: ADF layers

Above this layer, there are two important components: *ADF Agent* (see section 3.4.2) and *ADF Naming Service* (consult section 3.4.4).

The developer has no directly access to these layers, because he/she can use the *ADF API*. From architectural point of view, the ADF framework is encapsulated within an abstract agent classes.

Actual layered structure of the ADF system is presented in figure 5.

3.5 Results and Examples

3.5.1 Advantages and Disadvantages

We enumerate some of the advantages of our agent-oriented proposal:

- *A flexible API and a simple programming model*

  The ADF framework was designed to provide a straightforward programming model, helping the developer to easily create new applica-

tions using the API. The framework is distributed in packages, libraries that contain the interfaces, abstract classes to be used by the developer. The proper documentation of ADF API is also available. The ADF package can be used in conjunction to a visual developing environment (see section 3.5.2).

- *Support for messaging*

  The standard API supports messaging between agents and/or hosts. The messaging support has been provided by the JMS package from Java.

- *Ability to migrate agents*

  This feature is also provided in the standard API, although the developer has the role to use the migration support.

- *Built using current standards*

  The ADF framework conforms to existing standards (some of them directly implemented by JDK and J2EE).

The base agent system infrastructure that can be used in software agent developing process is created. The ADF framework is free and can be easily updated, thus realizing a good and extensible support in development of agent-oriented applications.

One of the important feature – secure execution environment – is not still available in the current version and it will be addressed in near future.

3.5.2 Example

Following [1], we'll give an example of building an agent specialized in searching images on every host of a multi-agent system.

The agent has associated an *ImagesTask* 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.

```
import java.util.*;
/* next directives import ADF API components */
import adf.server.host.Host;
import adf.server.ns.NamingService;
import adf.client.agent.Agent;
import adf.client.IRegister;
```

```
public class ImagesAgent extends Agent { // built agent

public void runOnHost() {
  ImagesTask it = (ImagesTask)this.getCurrentTask();
  ImagesService is = (ImagesService) this.getCurrentHost().
    getServicesAccessService().getService(this,
      it.getTaskMetaData());

  if ( is == null ) { // no service of this kind on this host
    it.setCompleted(false);
    return; // next step will be the agent migration to a new host
            // which matches the requested task
  }
  else {
    try {
      is.solveUsing(is);
      is.setCompleted(true);
    } catch ( TaskException te ) { // exception occurred
        // while dealing with the specified service
    }
  }
}
}

public class ImagesTask extends Task { // searh task
  public void solveUsing( Service s ) throws TaskException {
    results = s.runService(this.getTaskMetaData());
  }
}
```

We can remark that it is not difficult to use the proposed API. 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 $\langle$ *task_type = search, content = image, type = JPEG, metadata = 'sunset forest'* $\rangle$.

The structure of above code can be also used in other situations, without modifications of the depicted classes.

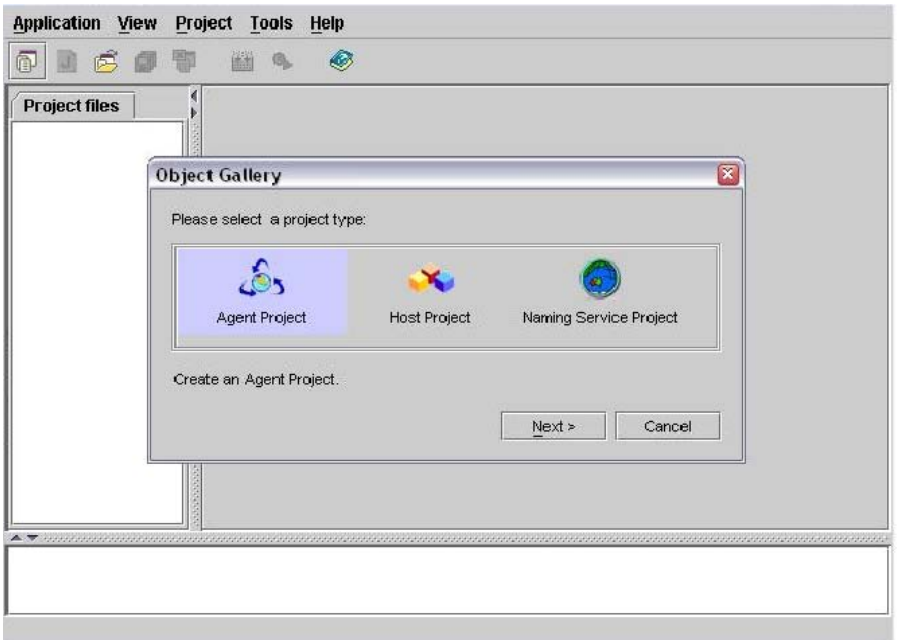

Figure 6: Choosing the project type

The implementation of the agents and the whole multi-agent system can be easily accomplished using a visual tool written in Java, a companion of ADF API. Our solution is able to automatically generate the essential structure of the Java source-code. The developer's task is to implement only the abstract methods provided by ADF.

First of all, the programmer must choose the type of the desired project (see figure 6). Currently, three project solutions are available: *Agent*, *Host* and *Naming Service*. The graphical environment is flexible enough to incorporate other projects (solutions), without recompiling the application.

The generated code for all involved classes is shown in figure 7.

More details regarding the current status of the ADF project are available at `http://www.infoiasi.ro/~busaco/projects/`.

4 Conclusion and Further Work

The paper presented ADF – an abstract framework designed and built with the purpose of developing agent-oriented applications, such as mobile agents or multi-agent systems. The provided API, written in Java and detailed in section 3.4.1, can be easily extended and used, in order to assure scalability and portability. The package that contains the agent framework can be used

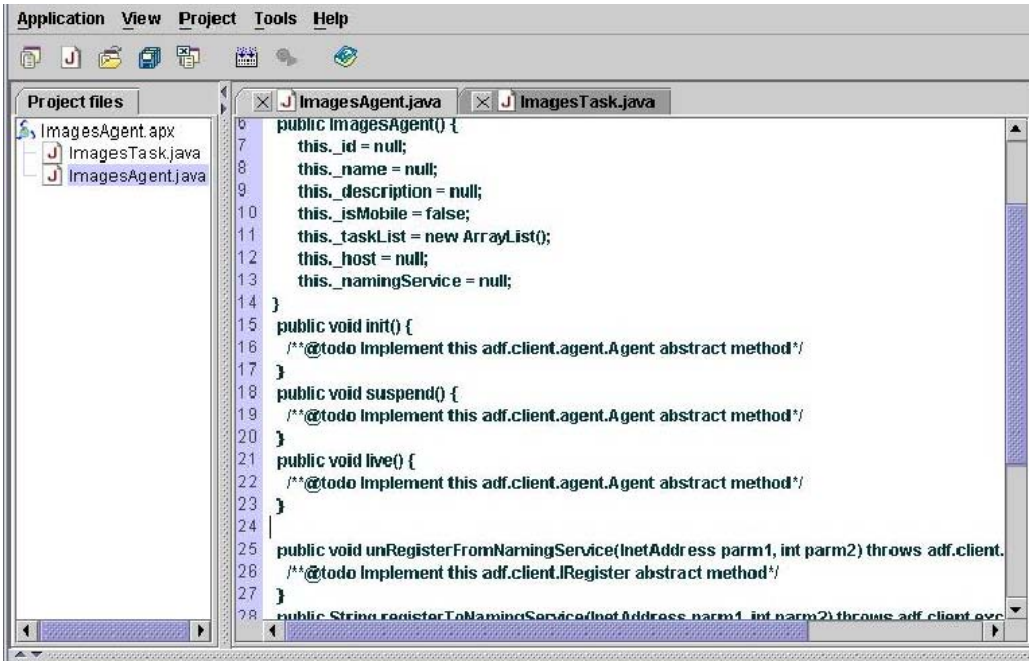

Figure 7: Browsing the generated code used to implement *ImagesAgent*

as a plug-in and can be deployed by a more complex application, such as an existing Java visual developing environment or an agent-based programming tool. With certain modifications, the provided API can be used – via J2ME (Java 2 Micro Edition) [20] – in the context of small mobile devices.

The example detailed in section 3.5.2 proves that ADF can offer support for Grid-like architectures, since the activity of searching data can be accomplished by agents. This aspect opens an interesting perspective in using ADF to build the application layer of the *tuBiG* [1] system, following the directions expressed in [7] and [16].

Another possible approaches are to integrate ADF within FIPA-OS framework and to align ADF project to semantic Web directions [10] – for example, to provide metadata and ontological support, describing agents, hosts and their interactions in OWL (Web Ontology Language) [25] or OWL-based languages.

Acknowledgement

We express our gratitude to *Professor Dorel Lucanu* (Faculty of Computer Science, "A. I. Cuza" University of Iaşi, Romania) for his useful remarks regarding the preliminary version of this technical report.

References

[1] L. Alboaie, S. Buraga, S. Alboaie, "*tuBiG* – A Layered Infrastructure to Provide Support for Grid Functionalities", in *Proceedings of the 2nd International Symposium on Parallel and Distributed Computing*, IEEE Computer Society Press, 2003

[2] S. Alboaie, S. Buraga, L. Alboaie, "An XML-based Object-Oriented Infrastructure for Developing Software Agents", *Scientific Annals of the "A. I. Cuza" University of Iaşi – Computer Science series*, tome XII, "A. I. Cuza" University Press, Iaşi, 2002

[3] S. Alboaie, S. Buraga, L. Alboaie, "An XML-based Serialization of Information Exchanged by Software Agents", *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics – SCI 2003*, Orlando, USA, 2003

[4] K. Blixt, R. Öberg, "Software Agent Framework Technologies", *Technical Report LiTH-IDA-Ex-00/14*, Linköpings Universitet, Sweden, 2000

[5] G. Booch, J. Rumbaugh, I. Iacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999

[6] J. Bradshow, *Software Agents*, AAAI Press, 1997

[7] S. Buraga, *Semantic Web* (in Romanian), Matrix Rom, Bucharest, 2004

[8] S. Buraga, G. Ciobanu, "A RDF-based Model for Expressing Spatio-Temporal Relations Between Web Sites", in *Proceedings of the 3rd International Conference on Web Information Systems Engineering – WISE, IEEE Computer Society Press*, 2002

[9] S. Buraga, S. Alboaie, L. Alboaie, "An XML/RDF-based Proposal to Exchange Information within a Multi-Agent System", in *Proceedings of NATO Advanced Research Workshop on Concurrent Information Processing and Computing*, IOS Press, 2004 – to appear

[10] J. Davies, D. Fensel, F. van Harmelen (eds.), *Towards the Semantic Web*, John Wiley & Sons, England, 2003

[11] S. Green, F. Somers, *Software Agents: A Review*: `http://www.cs.tcd.ie/research_groups/aig/iag/iag.html`

[12] D. Lange, M. Oshima, *Mobile Agents with Java: The Aglet API*: `http://www.comp.nus.edu.sg/~cs4274/wwwj.pdf`

[13] M. Luck, P. McBurney, C. Preist, *Agent Technology: Enabling Next Generation Computing*, AgentLink, 2003: `http://www.agentlink.org/`

[14] E. Mangina, *Review of Software Products for Multi-Agent Systems*, AgentLink, 2002: `http://www.agentlink.org/`

[15] D. Milojicic (ed.), "Mobile Agent Applications", *IEEE Journal on Concurrency*, July-September 1999

[16] O. F. Rana, L. Moreau, "Issues in Building Agent-based Computational Grids", *Third Workshop of the UK Special Interest Group on Multi-Agent Systems – UKMAS 2000*, Oxford, UK, 2000

[17] A. R. Tripathi *et al.*, "Design of the Ajanta System for Mobile Agent Programming", *Journal of Systems and Software*, May 2002

[18] M. Wooldridge, N. Jennings, "Pitfalls of Agent-Oriented Development", *Proceedings of the 2nd International Conference on Autonomous Agents*, ACM Press, 1998

[19] * * *, *AgentWeb*: `http://www.cs.umbc.edu/agents`

[20] * * *, *Java Software*: `http://www.javasoft.com/`

[21] * * *, *The Foundation of Intelligent Physical Agents*: `http://www.fipa.org/`

[22] * * *, *The Foundation of Intelligent Physical Agents – Open Source*: `http://fipa-os.sourceforge.net/index.htm`

[23] * * *, *Tryllian's ADK (Agent Development Kit)*: `http://www.tryllian.com/sub_company/software.shtml`

[24] * * *, *Voyager*: `http://www.objectspace.com/voyager`

[25] * * *, *World Wide Consortium's Technical Reports*, Boston, 2004: `http://www.w3.org/TR/`