A Pattern Language for FIPA Agent Interface Design

Vassilis Konstandinidis, Anthony Karageorgos

***Abstract* — This paper presents a pattern language supporting the design of web/mobile interfaces to static agents built by FIPA-compliant agent toolkits.**

The approach followed was to collaboratively gather the requirements the language should satisfy, and derive, select and combine appropriate patterns fulfilling them.

The use of the language has been validated in four case studies and its suitability for meeting the agent interface development requirements has been evaluated using Force Resolution Maps (FRMs). It is has been found that the proposed language fully satisfies the identified requirements.

***Keywords* — Agent Interfaces, Design Patterns, FIPA, Pattern Languages.**

I. INTRODUCTION

Despite the emergence of numerous methodologies for engineering agent systems there is no proportional development of systematic techniques for designing agent interfaces enabling agent access over the web and from mobile devices. To address this issue, in this paper we propose a design pattern language suitable for the development of web/mobile interfaces to static agents built by FIPA-compliant agent toolkits. The selection of the patterns and rules comprising the language was driven by a set of requirements that targeted agent systems should satisfy, which was identified in a collaborative manner based on experiences in developing web/mobile and agent applications.

The validity of the proposed pattern language is demonstrated in four different agent applications each focusing on different language aspects. Furthermore, the language is evaluated using Force Resolution Maps (FRMs) [1]. FRMs have been used to evaluate the language against its forces and against other design patterns that are similar to the ones used in the language.

In Section II we discuss the motivation for developing a pattern language for agent interfaces, followed by a description of the requirements such a language should meet in Section III. The proposed pattern language is presented in Section IV and some applications developed to test it are discussed in Section V. The suitability of the proposed language for designing agent interfaces is evaluated in Section VI, and relevant work is presented in Section VII. Finally, Section VIII concludes the paper.

Vassilis Konstandinidis is an IEEE member and works for Byte S.A, Athens 117 41, Greece (email: V.Konstandinidis@computer.org)

Antony Karageorgos is an IEEE member and works for the department of Communications & Computer Engineering, University of Thessaly, Volos 38 221, Greece (email: karageorgos@computer.org)

II. MOTIVATION

To address the complexity inherent in agent design and to facilitate interoperation of heterogeneous agent systems developed using different tools and methodologies various aspects of agent systems have been standardized by organizations such as the IEEE standards committee FIPA [7]. Based on these standards, numerous agent toolkits have appeared ([2]-[6]) both under commercial and open source licences. A common aspect in all these toolkits is that they provide limited support to agent developers for designing remote, i.e. accessible from another location using internet or mobile device technology, interfaces for agents in a systematic and effective way.

Design patterns can greatly facilitate the design of software systems in a plethora of ways [8]. However, despite their benefits individual design patterns alone are not sufficient for building complete/complex software systems. As Coplien [9] emphasises *"A pattern in isolation solves an isolated design problem; a pattern language builds a system. It is through pattern languages that patterns achieve their fullest power"*.

Therefore, it came natural that the problem of designing web-based and mobile agent interfaces can be resolved by using a suitable pattern language targeting the interface requirements of agent applications, which is the focus of the work described in this paper.

III. REQUIREMENTS OF THE TARGETED SOFTWARE SYSTEM

A. Requirements gathering process

To determine the requirements of the targeted agent software interfaces we held a series of Joint Agent Development (JAD) [10] sessions, during which we divided the interested parties into three groups: End users, members of development teams that deal with the agent part of the targeted system, and members of development teams that deal with the OO part of the targeted system.

B. Requirements

The results of JAD sessions are summarised in the following requirements that agent interfaces must satisfy:

- The presentation and command aspects of the web/mobile interface application must be decoupled from its conceptual aspects.
- The web/mobile interface part of the system need contact the agent via method calls (keeping in mind that an agent communicates with the outside world through message calls).

- The entire system must be distributed, where perhaps all its components sit on different machines and communicate to each other through remote method calls.

IV. THE PATTERN LANGUAGE

A. Prerequisites

As with all pattern languages, a list of pre-conditions must hold before this pattern language can be implemented. In this case, the pre-conditions are two:

- First, there needs to be an agent platform ready that will host the targeted agent (or agents, if the pattern language is implemented more than once to target more than one agent).
- Secondly, the targeted agent(s) must be up and running and their APIs known to the developer(s) of the pattern language.

B. Individual patterns

Having studied a plethora of design patterns and from a variety of sources we concluded that the following patterns should be included in the pattern language to address the requirements/problems presented above:

- Model-View-Controller (MVC) [8] to address the first force.
- Adapter [11] to address the second force.
- Broker [8] to address the third force.

The reasons for this choice can be summed as follows:

a. During the JAD sessions it was made clear that the targeted system can be constructed solely on techniques and technologies that belong to the OO world (with the exception of the agents themselves, of course). More precisely, given the API of the agent(s) we want to create a web/mobile interface for, the rest of the system can be constructed in a way that is no different from any other web/mobile application. Hence, the three patterns in the pattern language are OO design patterns and not agent-related patterns.

b. The Adapter, MVC and Broker patterns are very popular and well-documented patterns that many developers/systems analysts know how to use/implement in different programming languages.

C. Domain of the Pattern Language

The language proposed here can be classified as *"targeting agent based systems, which are built solely on FIPA-compliant toolkits, where all the agents involved are static (not mobile)."*

D. Representation

The pattern language can be represented graphically as follows:

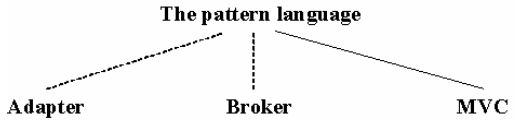

Figure 1: The structure of the pattern language

As we can see from the figure, the Adapter and Broker patterns need to be implemented before the MVC pattern. The role of the Adapter is to pass messages from the web/mobile interface to the agent, in a format that the agent application understands and vice versa. But the Adapter might not be running on the same machine as the user interface application. How can the user application get a reference to the Adapter? The answer is to implement the Broker pattern for the Adapter (i.e. the Adapter will be the service of the Broker architecture). It does not really matter which pattern to implement first, as the implementation of one does not interfere/depend on the implementation of the other.

E. Set of rules

Coplien has said that: *"A pattern language defines a collection of patterns and the rules to combine them into an architectural style."* [12].

Following this statement, we present a set of rules to combine the different patterns in the proposed pattern language in Table 1 below:

TABLE 1: SUMMARY OF THE RULES IN THE PATTERN LANGUAGE

Rule	**Apply when**
Create an Adapter for the agent, make it available across different Java Virtual Machines (JVMs), across a network or networks, and therefore to the user interface application, through the use of a Broker, and develop the user interface application itself using the MVC pattern, in the following order: Adapter or Broker followed by MVC.	- The agent and the Adapter run on a machine A, and the user interface application runs on a machine B. - The agent, the Adapter and the user interface application are all hosted on the same machine. - The Adapter and the user interface application run on a machine A, and the agent runs on a machine B. - The agent, the Adapter and the user interface application all run on different machines.
Use just the MVC pattern to create the user interface application itself. There is no need to implement other patterns to access the agent from the UI application.	The agent toolkit used is LEAP running on a pJava or MIDP device, we want to create a mobile interface to the agent running on that device, and we want to create this interface on the same device.

F. Variants

In the case where the agent toolkit used is LEAP running either on a pJava or MIDP device, the pattern language consists only of the MVC design pattern.

V. TESTING

A. Introduction

In order to validate the pattern language we used it to design four different agent-based applications. Demo applications I, II and III are direct implementations of the pattern language, whilst demo application IV is an implementation of its variant.

B. Demo application I

1) Introduction

In this demo application we created a web interface to an open-source, FIPA-compliant agent platform that: a) displays a list of all the agent platforms that are up on the agentcities.net network [13], b) allows the user to select a platform from that list and send to it a simple "ping"

message, and finally c) get a reply from the remote platform.

In creating this application we set up the UMIST Agent City [14] and joined it to the agentcities.net network.

2) Structure

Following the pattern language, the structure of the system looks as follows:

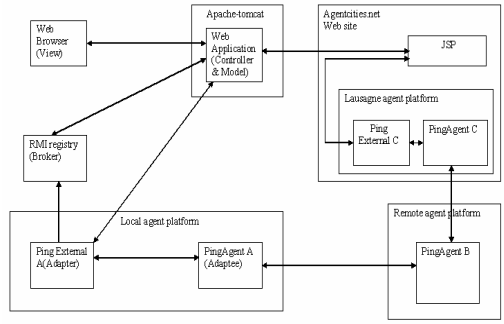

Figure 2: Structure of demo application I

C. Demo application II

1) Introduction

This demo application is actually based on the "FruitMarket" demo that comes with the distribution of ZEUS, where instead of simply creating AWT interfaces for the "ShopBot" and "SupplyBot" agents we created Java applet interfaces for both agents.

2) Structure

Following the pattern language, the structure of the system looks as follows:

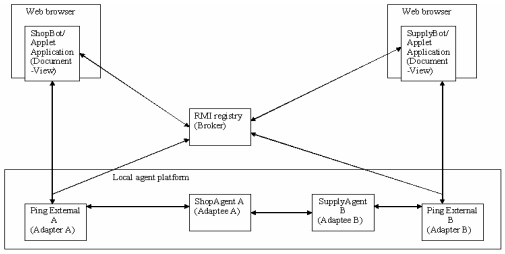

Figure 3: Structure of demo application II

D. Demo application III

1) Introduction

This is essentially the same application as the demo application we presented above with the only difference being that the interface instead of being web-based it is now a mobile-based one.

2) Structure

Following the pattern language, the structure of the system looks as follows:

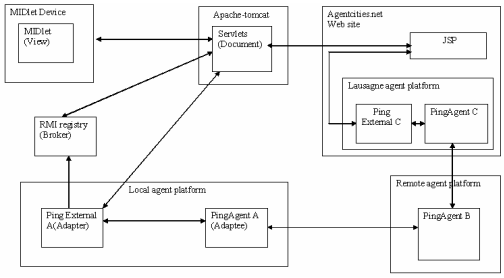

Figure 4: Structure of demo application III

E. Demo application IV

1) Introduction

In this demo application we created two different agents running on the mobile agent platform LEAP: a) one that is fired up when the LEAP main container starts and runs on the same PC as the LEAP main container, and b) another one that runs on a mobile device that runs a LEAP container (not the main). The first agent has an AWT interface that displays the number of messages that this agent sends to the second agent and vice versa. The second agent has an MIDP or Personal Java interface that does the same thing.

2) Structure

Following the variant of the pattern language, the structure of the system looks as follows:

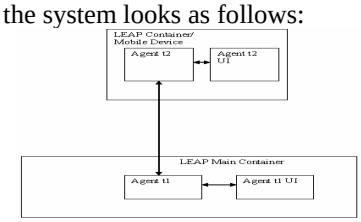

Figure 5: Structure of demo application IV

VI. EVALUATION

A. Meeting the pattern language forces

Based on the results of the demo applications we developed to validate the pattern language, we can safely say that it successfully addresses its forces. Since this alone is not enough to completely conclude about its quality we compare it to other solutions in the next section.

B. Comparison with other solutions

In this section, we use FRMs [1] to validate the pattern language against a number of forces relevant to agent interface development. FRMs can help designers decide if a pattern fits the application requirements and can possibly be applied, and if yes whether it is the best candidate.

The following forces are considered: Real-time responsiveness, Reliability, Data management security, Complexity, Efficiency, Recovery, Maintainability, Accuracy, Effectiveness, Reusability, Need to decouple the presentation/UI of the agent from its functional core, Need to convert the interface of the targeted agent into another interface that the non-agent, UI, client application can use to contact the agent, Need to make the system distributed when the UI application is running on a different machine than that of the agent.

C. Adapter pattern FRM

Starting with the Adapter pattern we compared it against the Bridge, Decorator, Proxy and Strategy patterns [8] and we concluded to the following FRM:

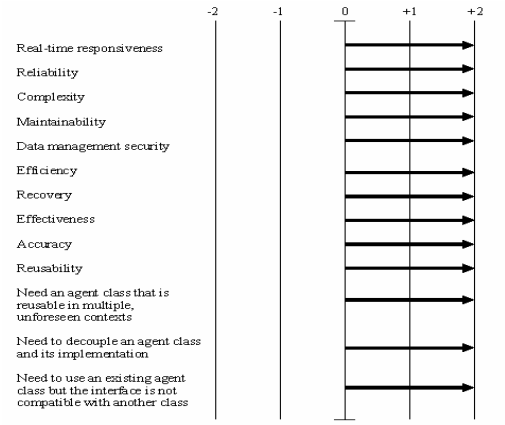

Figure 6: An FRM for the Adapter pattern

D. MVC pattern FRM

On with the MVC pattern we compared it against the PAC and View Handler patterns [11] and we concluded to the following FRM:

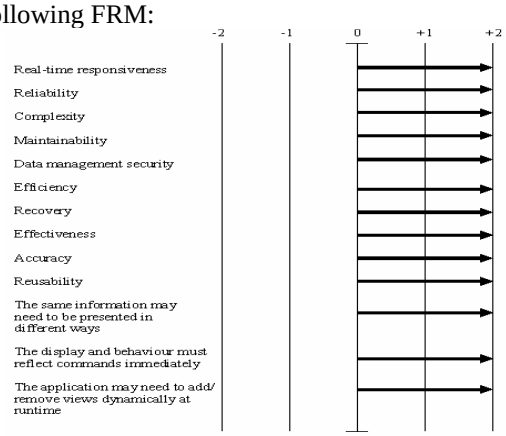

Figure 7: An FRM for the MVC pattern

E. Broker pattern FRM

Finally, in the case of the Broker pattern we compared it against the Forwarder-Receiver, Proxy, Client-Dispatcher-Server, Mediator and Lookup patterns [11] and we concluded to the following FRM:

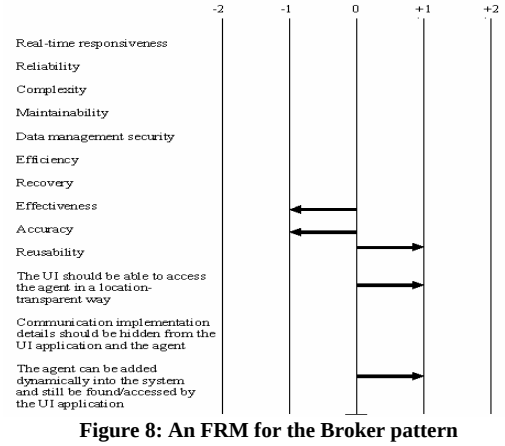

Figure 8: An FRM for the Broker pattern

F. Pattern Language FRM

Combining all the above FRMs together we concluded to the following FRM for the pattern language:

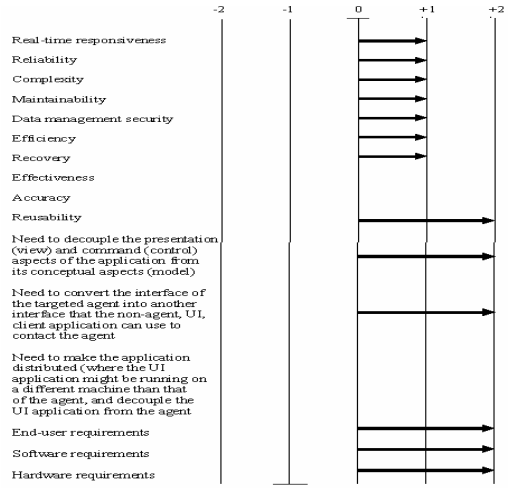

Figure 9: An FRM for the pattern language

The results of this FRM show that the pattern language resolves most of its forces in a positive way. Three of its forces though are resolved moderately well, and more work should be done to change this.

VII. RELEVANT WORK

Current agent development toolkits do not provide adequate support for building agent interfaces. The solution given by JADE [2] is unreliable targeting only JSP-based web interfaces. LEAP [3] and MicroFIPA-OS [4] do not consider agent interaction between different mobile agent platforms, and DialoX [15], the solution given in April toolkit [5] does not consider agent mobility.

VIII. CONCLUSIONS AND FURTHER WORK

We proposed a pattern language supporting the design of agent interfaces. We considered static agents and FIPA compliant agent toolkits. We showed that the language meets satisfactorily a set of necessary requirements. Future work includes extending the language with additional patterns and with guidelines for their proper use.

REFERENCES

[1] J. Souza, S. Matwin, N. Japkowicz, "Evaluating Data Mining Models: A Pattern Language," *Proceedings of the 9th Conference on Pattern Language of Programs (PLOP'2002)*, 2002.
[2] http://sharon.cselt.it/projects/jade (Accessed: 15 June 2005).
[3] http://leap.crm-paris.com/index.html (Accessed: 15 June 2005).
[4] http://www.emorphia.com/research/about.htm (Accessed: 15 June 2005).
[5] http://sf.us.agentcities.net/aap/ (Accessed: 10 March 2005).
[6] http://sourceforge.net/projects/zeusagent/ (Accessed: 10 March 2005).
[7] http://www.fipa.org (Accessed: 15 June 2005).
[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
[9] J. O. Coplien, "*Software Patterns*", SIGS Books, 1996.
[10] Jane Wood, Denise Silver, *"Joint Application Development"*, 2nd ed., New York : Wiley, 1995.
[11] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, *A System of Patterns: Pattern-Oriented Software Architecture*, Wiley, 1996.
[12] http://hillside.net/patterns/definition.html (Accessed: 15 June 2005)
[13] http://www.agentcities.net (Accessed: 15 June 2005).
[14] http://www.agentcities.co.umist.ac.uk/ (Accessed: 15 June 2005)
[15] http://sourceforge.net/projects/networkagent/ (Accessed: 10 March 2004).