Lightweight Communication Platform for Heterogeneous Multi-context Systems: A Preliminary Report [⋆]

Vladimír Dziuban, Michal Čertický, Jozef Šiška, and Michal Vince

Department of Applied Informatics, Faculty of Mathematics, Physics and Informatics
Comenius University, Bratislava,
Slovakia`{dziuban,certicky,siska,vince}@ii.fmph.uniba.sk`

Abstract. This paper describes an ongoing implementation of a lightweight communication platform that uses RESTful TCP/IP requests to transfer FIPA ACL based messages between agents. The platform is intended for heterogeneous systems composed of numbers of simple agents as opposed to usual FIPA implementations. Agents can be written in any language and can communicate with each other without the need for a central platform.

1 Introduction

Multi-context systems describe information from different viewpoints (contexts) and the relationship between them in the form of bridge rules. Heterogeneous multi-context systems [1] allow different logics or completely different formalisms to be used in different contexts.

When building truly heterogeneous multi-context systems in a decentralised fashion, components built on different formalism can be implemented in different languages and must cooperate and communicate with each other. FIPA [5] defines standards for such interoperability of heterogeneous agents. However most implementations involve complex, mostly java-based, platforms, usually hosting multiple agents. To facilitate simpler and more agile development of small agents in different languages, a more lightweight approach is needed.

We present an ongoing implementation of a de-centralised lightweight communication framework for heterogeneous agents that implements a subset of FIPA specifications, while allowing simple development of small standalone agents. Agents in the framework are standalone processes, that communicate using FIPA ACL messages[2], transported through RESTful[7], peer-to-peer TCP/IP connections.

There is no full featured platform with its associated services. However, the framework provides a simple agent management system [3] in the form of a

[⋆] We acknowledge support from VEGA project of Slovak Ministry of Education and Slovak Academy of Sciences no. 1/0688/10 and Comenius University grant no. UK/486/2010.

discovery service that allows each agent to identify other agents on the local network along with the services they provide. This allows creation of systems consisting of multitude of small heterogeneous agents scattered through the local network without the need for a central server / registry. Communication with agents on remote networks or with other FIPA platforms / implementations with different message transport systems can be achieved through the use of gateway agents that forward messages and agent information. Implementations in Python and C++ are available and Java implementation is planned.

Example 1. Consider a heterogeneous multi-context system used to organize seminars of a research group. It consists of three types of agents:

– *Personal* Java-based agents running on the mobile phone/PDA of every group member, providing information about him and his schedule,
– *sensoric* C++ agents that use webcameras to observe certain rooms at the university,
– *timetable* agent that provides access to the university timetable and room reservations,
– and a logic based *scheduling* agent (written in Python as a frontend to a logic based formalism such as an ASP solver.)

Now imagine, that one of the group members wants to organize a meeting with his colleagues. He simply orders his *personal agent* to arrange this meeting. This agent then contacts all the other *personal agents*, and finds out (using the GPS on their devices) which of them are currently out of town. If the sufficient number of them are available, he asks the *scheduling agent* for the most appropriate time and place/room for the meeting. The *scheduling agent* collects information from the *sensoric agents* to find out which rooms are empty and also checks if they aren't reserved for the next 2 hours. After receiving this information, the invitation can be sent to all the *personal agents* of available members of the group.

The rest of this paper is structured as follows: in the next section we describe the presented communication platform; the third section describes our implementation; the fourth section contains a short comparison to other FIPA platforms; and the last section presents future plans and concluding remarks.

2 Communication Platform

This section describes a lightweight communication platform. Such a platform consists of *agents* that can send *messages* to each other. Agents are usually standalone processes / programs and can run on different computers. Messages conform to FIPA ACL specification[2] and are transported through a stateless TCP/IP connection to the recipient.

Each agent has a globally unique *agent name* that is used to identify the agent and a list of *services* that the agent provides.

An agent has access to two basic services: discovery service that serves as a very simple AMS and a message transport service. These are implemented per process/program and are thus shared between multiple agents running in a single process.

Discovery service monitors the local network and notifies the agent of the appearance or disappearance of other agents. It maintains a list of known agents and their locators.

2.1 Message Transport Service

A transport service with a single protocol is used. The protocol uses RESTful TCP/IP connections to another agent (not necessarily the final recipient of the message) to deliver requests. There are two types of requests defined: a POST request that delivers a single message, or a POLL requests, that ask the other agent for any pending messages for the connecting agent.

The *transport specific address* of an agent is an IP address and port tuple and there are two transport-specific properties: polling mode and level of indirection.

Level of indirection represents how many times would the message be forwarded, would it be sent to this address. Each agent that acts as a proxy/gateway for another increases this number when it announces the target agent on a local network. If the discovery service reports multiple addresses for a single agent, one of those with lowest level of indirection should be used when sending messages.

If the polling mode is enabled for an address, then the message should not be send, but kept in a queue at the sending agent until the receiving agent asks for messages with the POLL request or the message expires (either through a pre-set timeout or because of a size limit of the queue.)

This can be used by agents that don't have a stable or accessible address on the network, such as on a mobile device that roams between different network connections. Such an agent would normally register with a gateway agent, which would announce it on the local network, collecting all messages and delivering them later through the connection created by the first agent's POLL requests.

2.2 Gateway agents

A discovery service, as described in the previous section, can reliably work only on a local network. Similarly the message transport can deliver messages only to agents on the local network or with a publicly accessible IP address (e.g. not behind NAT.) These restrictions can be worked around by the introduction of special *gateway agents*.

A gateway agent (GA) is a special agent that acts as a proxy for agents from other networks. GA maintains a list of registered remote agents. When a remote agent registers, GA announces remote agent's presence on the local network with its own transport specific address. Thus any message sent to the remote agent from the local network is sent to GA, which looks up the remote agent in its database and delivers (forwards) it.

There are two basic ways to use gateway agents:

- remote agents register directly with gateway agents
- a *bridge* agent registers all agents on his local network with a gateway agent on a remote network (and vice versa).

Similarly, a gateway agent that acts as a bridge to other message transport systems can be created, thus enabling interoperability with other FIPA compliant platforms.

3 Implementation

This section presents the implementation of the communication platform.

The aim of our implementation is a simple and lightweight framework, that can be used also on devices with little memory and computational power, e.g. cell pohones, embedded devices, etc.

Our architecture is currently implemented in Python and C++, which were selected for their simplicity and speed respectively. In the future we plan to provide an implementation in JAVA, as a language most commonly associated with multi-agent system development, that might encourage further development of heterogenous agent applications. In the current implementation, single or multiple instances of the agent can be used per process, being served by the same discovery and message transport service

In the discovery service, multicast protocol (there are also plans to add Avahi/Zeroconf support) is used to announce the arrival of a new agent to the network. Following registration routine is then handled by message transport service by sending a request message in ACL to the new agent and inform message with agent's properties as an answer to this request.

Message transport service is responsible for marshalling and demarshalling messages, sending and receiving over TCP/IP protocol and posting them to the main loop of the corresponding agent.

Agent's mainloop is event driven, with four main types of events:

- agentAdded triggered when a new agent registers a service
- agentRemoved triggered when the agent leaves the network
- agentChanged triggered when the agent changes his serivces
- messageReceived triggered when message is reveived.

MessageReceived is then further marshalled by communicative act[4] to InformMessage, RequestMessage, QueryIfMessage, etc or to SystemMessage.

We are currently working on the implementation of gateway agents.

4 Comparison to Other Work

There are many different FIPA compliant frameworks, most of them are implemented in JAVA and their platforms offer many services. This makes them computationally and memory intensive, therefore running them on small devices with little memory and slow processors is very difficult, often even impossible.

Our solution does not implement agent platform or platform services. The latter can be however implemented in form of standalone agents. This approach is needed to ensure, that agents are more lightweight and can be deployed easily and with as few preliminary arrangements as possible.

SPADE[8, 6] might be an example of a more lightweight approach, although SPADE agents run on a platform with standard FIPA AMS and DF components. It is written in Python and uses XMPP protocol for message transportation. Each agent is a client with a registered Jabber ID and all communication is conducted by sending Jabber messages that contain FIPA ACL expressions. The platform however requires a central jabber/XMPP server.

5 Conclusion and Future Work

In this paper we have presented a lightweight communication platform designed especially to allow easy implementation of heterogeneous multi-context systems. The platform uses FIPA ACL messages transported through simple RESTful peer to peer TCP/IP connections. Each agent also has a discovery service that acts as a simple agent management system.

The platform also allows the creation of gateway agents that can be used to interconnect agents from different networks or based on other FIPA platforms using different message transports.

Each agent keeps a local agent directory via his discovery service, which might not scale well for larger systems. Gateway agents could take the role of directory services, especially since they can be dynamically created/registered. Ordinary agents will use local network discovery services only to find gateway agents and use them as a full featured agent management system/directory facilitator.

A possible improvement of the dynamic aspects of the platform is the ability to preserve its structural integrity by automatic reorganization of local-area components. Basic idea behind this is making every agent capable of starting a gateway service whenever one is needed (i.e. when the number of local gateway agents is critically low) and obtaining the needed database from other remaining local gateway agents. This makes it possible to maintain the connectivity with external networks or platforms even after the loss of several gateway agents.

References

1. Brewka, G., Eiter, T.: Equilibria in heterogeneous nonmonotonic multi-context systems. In: AAAI. pp. 385–390. AAAI Press (2007)
2. Foundation for Intelligent Physical Agents: FIPA ACL Message Structure Specification. http://www.fipa.org/specs/fipa00061/index.html (2000)
3. Foundation for Intelligent Physical Agents: FIPA Agent Management Specification. http://www.fipa.org/specs/fipa00023/index.html (2000)
4. Foundation for Intelligent Physical Agents: FIPA Communicative Act Library Specification. http://www.fipa.org/specs/fipa00037/index.html (2000)
5. Foundation for Intelligent Physical Agents: FIPA Standard Specification. http://www.fipa.org/repository/standardspecs.html (2000)

6. Gregori, M.E., Cámara, J.P., Bada, G.A.: A jabber-based multi-agent system platform. In: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. pp. 1282–1284. AAMAS '06, ACM, New York, NY, USA (2006), http://doi.acm.org/10.1145/1160633.1160866
7. Representational state transfer. http://en.wikipedia.org/wiki/Representational_State_Transfer
8. Spade2: Smart python agent development environment. http://code.google.com/p/spade2/