LCP: A Lightweight Communication Platform for Heterogeneous Multi-context Systems [*]

Michal Čertický, Jozef Šiška, and Michal Vince

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

Abstract. This paper describes an 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 distributed 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 Heterogeneous Multi-Context Systems

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 for various logics or completely different formalisms to be used in individual contexts. Intended applications of multi-context systems thus include highly heterogeneous and distributed systems with components using different formalisms, implemented in various languages, and for wide variety of devices.

Foundation for Intelligent Physical Agents (FIPA) [5] defines standards for interoperability of such systems. There are many different FIPA compliant frameworks, most of them complex platforms (usually) implemented in JAVA. They offer many services which makes them computationally and memory intensive, therefore ill suited for truly heterogeneous system and not feasible for small devices with constrained memory and processor power. To facilitate simpler and more agile development of small agents in different languages, a more lightweight approach is needed.

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

– *Personal* JAVA-based agents running on the mobile phone/PDA of every group member, providing information about him and his schedule,

[*] We acknowledge support from VEGA project of Slovak Ministry of Education and Slovak Academy of Sciences no. 1/0688/10.

– *sensoric* C++ agents that use webcameras and other sensors 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.)

Usually, the *personal* as well as the *sensoric* agents need to be implemented as simple and small programs/scripts running on mobile phones or embedded devices. Common way of incorporating such agents into FIPA-compliant platforms would be creating *proxy agents* inside the platform. These would communicate with those external personal agents which is not really a distributed solution.

2 Lightweight Communication Platform for Heterogeneous Systems

We present an ongoing implementation of a decentralised 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. Currently Python and C++ implementations are available and a JAVA version is planned. The implementation can be downloaded, along with related publications, at [6].

There is no full featured platform with its associated services. However, the framework provides a simple and automated agent management system [3] in the form of a 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 network without any need for a central server / registry.

Communication with agents on remote networks or with other FIPA implementations with different message transport systems can be achieved through the use of *gateway agents* that forward messages and agent information.

Each agent within a platform is identified by a globally unique *agent name* and a list of *services* that the agent provides. Every agent has access to two basic core 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 simply 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. *Message transport* service, on the other hand, is a bit more complex:

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 and a POLL request 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 along with 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. It is increased by each proxy/gateway agent when announcing remote agents. If the discovery service reports multiple addresses for a single agent, an address with the lowest level of indirection is used when sending messages.

Polling mode disables the delivery of messages but instead allows agents to ask (poll) for messages through POLL request. This can be used by agent without a stable or accessible address, such as roaming mobile devices.

2.2 Gateway agents

The discovery service works 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. These restrictions can be worked around through the use 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 its presence on the local network with GA's 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 either delivers (forwards) it or stores them to be later retrieved by the remote agent through a POLL request. A special kind of GA (a bridge) can automatically register all agents on his network with a GA on a remote network.

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.

2.3 Implementation Details

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 phones, embedded devices, etc.

As already mentioned, our architecture is currently implemented in Python and C++. They were selected for their simplicity and speed respectively. We are currently working on the implementation of gateway agents. In the future we plan to provide an implementation in JAVA, as it is a language most commonly associated with multi-agent system development, that might encourage further development of heterogeneous 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

Multicast protocol is used in the discovery module to announce the arrival of a new agent to the network, although there are also plans to add Avahi/Zeroconf support.

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.

Each agent runs an event driven mainloop, with four main types of events: agentAdded, agentRemoved, agentChanged and messageReceived. MessageReceived is then further marshalled by communicative act [4] to InformMessage, RequestMessage, QueryIfMessage, etc or to SystemMessage.

3 Conclusion

Our solution does not implement a full platform or platform services as defined by FIPA. 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.

Each agent in our platform keeps a local agent directory via his discovery service, which might not scale well for larger systems. To avoid that, gateway agents could take the role of directory services, especially since they can be dynamically created/registered. Ordinary agents could 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. We intend to make every agent a potential GA, and let the network self-organize. For example, if some of the agents disappear from the local network, and the number of GAs is dangerously low, some of the remaining agents would start acting like gateways in addition to their normal functions. This aims to maintain good connectivity between individual local-area parts of the system.

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. Lightweight communication platform. `http://ii.fmph.uniba.sk/~siska/lcp/`
7. Representational state transfer. `http://en.wikipedia.org/wiki/Representational_State_Transfer`