jfipa - an Architecture for Agent-based Grid Computing

Amund Tveit[*]
Department of Computer and Information Science
Norwegian University of Science and Technology

December 15, 2001

Abstract

With the increasing focus on grid development, there is a need for proper abstractions for modelling grid applications. Viewed from a distributed AI perspective the most suitable abstraction is the concept of agents. In this paper an agent-based architecture for grid computing is considered. The architecture enables routing and handling of FIPA ACL messages.

1 Introduction

With the increasing importance and potential of the Internet as an efficient global communication and computing infrastructure, the development and popularity of concepts and technologies related to *grid computing* has accelerated. The grid has been described as *"coordinated resource sharing in dynamic, multi-institutional virtual organizations"* [5]. These virtual organizations (VO) and their members are interconnected by a network, typically the Internet. Some examples of resource sharing are cpu-intensive processing (e.g. physical simulations), data storage and online services

If grid computing is considered from a *Distributed Artificial Intelligence* (DAI) perspective [9], it is clearly not a type of *Distributed Problem Solving* (DPS), since there is no or little central control in the grid. Its resemblance with a *Multi-Agent System* (MAS) is much higher, since entities in a MAS and the grid have autonomous behavior (i.e. distributed control).

Autonomy is one of the key abstraction features of agents [17]. Other features of agents relevant for grid entities include *social ability*, as well as *reactive* and *pro-active* intelligence. This justifies the selection of agents as the main abstraction for grid entities.

[*]amund.tveit@idi.ntnu.no

1.1 Grid Routing Issues

However, since one often talks about *the* grid, it means that the *whole* Internet and the VOs can be seen as the edges and nodes of a graph representing the grid. This graph is *very* large, and far from being complete, which means that a message sent between two nodes need to travel a path through intermediary nodes before arriving at its final destination. The process of finding this path is called a *routing algorithm.*

Three important metrics of an efficient routing algorithm are the number of intermediate nodes in a path, the path cost, and the delay [14]. They should all be minimized according to expression (1).

$$min\ (c_1 \cdot hops + c_2 \cdot pathcost + c_3 \cdot delay) \quad (1)$$

When messages are forwarded through the grid based on other criterias than the receiver address, e.g. service description or price, the process is commonly called *automatic brokering.* Even though these processes have new metrics dependent on the query (e.g. quality, cost or relevance of the result), the fundamental metrics of routing are still of importance.

1.2 Problem

The overall problem is: *how to enable efficient routing and brokering in an agent-based grid?*

More particularly, how to create an architecture: *1) that is pluggable with respect to routing algorithms, 2) has a simple and easy-to-use API, 3) has high processing and IO performance, and 4) has a robust implementation.*

The rest of this paper is organized as follows. Section 2 describes and discusses the architecture, section 3 describes the implementation, section 4 describes related work, and finally the conclusion with future work.

2 Architecture

2.1 Architectural Goals

The overall goal of the jfipa architecture is to become a simple, yet efficient and extensible, router architecture supporting agent-based grid computing. Agents should be able communicate using the speech act-based [13] FIPA Agent Communication Language (ACL). The primary supported encoding of the ACL should be XML, and the primary communication protocol (application-level) should be HTTP.

As the j in jfipa insinuates, java is the primary implementation language, but also other common programming languages will be able to use jfipa through a planned XML-RPC interface.

2.2 Scalability Goals and Approach

The jfipa architecture should be able to scale in several different dimensions: 1) increase in the agent identification address space [11] (related to future IPv6-based Internet), and 2) an increase in total number of agents [12] (residing on different jfipa-nodes) and messages. The primary approach of enabling this scalability is through efficient routing of ACL messages by supporting pluggable routing algorithms. Other scalability-enabling approaches (e.g. caching) is outside the scope of jfipa.

2.3 jfipa Overview

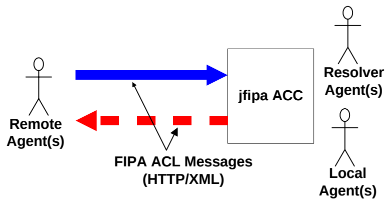

Figure 1: jfipa Overview

Figure 1 shows the three main components of the jfipa architecture. *Resolver agents* provide routing and brokering support for incoming ACL messages, and the *local agents* provide any type of service that is not related to routing or brokering. The *jfipa Agent Communication Channel* (ACC) provides support for communication, parsing and queueing of ACL messages, as well as administration of the resolver and local agents. *Remote agent(s)* are agents that access the jfipa architecture externally using FIPA ACL.

Note that the term *local agent* doesn't necessarily mean that the agent and the jfipa ACC reside on the same computer with the same network address, but it does mean that the local agents use the services of the communication channel, hence it doesn't need to have its own FIPA ACL handling mechanisms.

2.4 jfipa ACC Architecture

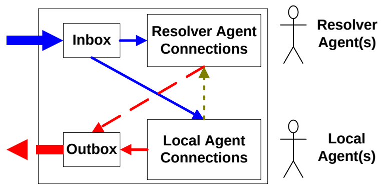

Figure 2: Agent Communication Channel

Figure 2 shows the four main components of the jfipa ACC. The *Inbox* is where incoming ACL messages are received (i.e. from remote agents), *Outbox* is where the internally generated or routed messages are being sent from (i.e. messages from local or resolver agents). *Resolver Agent Connections* and *Local Agent Connections* handles ACC communication for the resolver and local agents, respectively.

The reason for having the Inbox and the Outbox handling communications instead of the Resolver and Local agents themselves, is that logging and administration becomes simpler.

2.5 Inbox

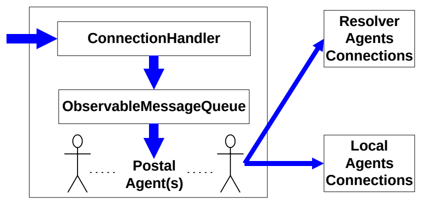

Figure 3: ACC Inbox

ConnectionHandler handles incoming network requests and inserts them into the *ObservableMessageQueue*, which notifies a *Postal Agent* about the new message. Connection between the ObservableMessageQueue and the Postal Agents are based on the *Observer Design Pattern* [6].

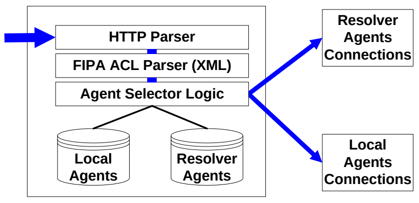

Figure 4: Postal Agent Architecture

2.6 Postal Agent Architecture

The postal agent extracts and parses the HTTP message its FIPA ACL XML-encoded envelope and message. It continues by checking if the message is addressed to a particular *local agent*, otherwise it passes on the message to the *resolver agents* for potential routing or brokering.

The Postal agents also handles the repository of registed local and resolver agents.

2.7 Agent Connections Architecture

The *Resolver Agents Connections* and *Local Agents Connections* architectures are very similar, except that the latter also supports forwarding of messages to the prior based on the results from local agent processing. Support for creating a large number[1] of (sophisticated) local agents is outside the scope of jfipa, and is supposed to be handled by other agent platforms that only use jfipa for external Fipa-based interaction (e.g. Agora [8]). Connections between jfipa and the other agent platform is done using two observer design patterns, where the local agent is the observer of the Incoming Messagehandler's queue and the observable for the Outgoing Messagehandler.

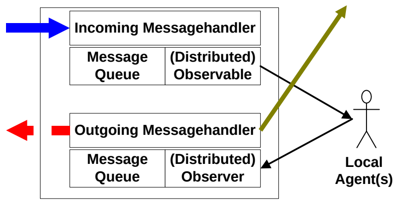

Figure 5: Agent Connections Architecture

In order to support interactions between few (non-sophisticated) local agents (e.g. wrapper agents for existing legacy systems), jfipa will use a hashtable of incoming/outgoing message queues where each hash key represents the name of a local agent, this ensures efficient ($O(1)$) sending of messages. The receiving agent gets notified about messages by using observing its message queue.

[1]more than 100 agents

2.8 Routing Issues

The main data structures for a routing algorithm are *routing tables*. Routing tables have information about the network topology, or in other words, knowledge about which direction (peer/neighbour) to best send/forward messages.

If the network is small (few nodes and edges) and relatively static (slow changes in topology), routing tables can efficiently being calculated using shortest path algorithms, e.g Bellman Ford's (BF) or Dijkstra's algorithm [2].

The main disadvantage of the shortest path algorithms is that they don't scale well with rapid increase in network size and changes in topology. They tend to be suboptimal because they are sending *all* traffic to the same destination through a *single* path (i.e. using only one peer per destination), instead of dividing the traffic among *several* paths [3]

The application of machine learning algorithms in adaptive routing, in particular reinforcement learning, has been shown to improve network throughput by up to three and a half times than routing with the BF algorithm [3]. The main reasons for the improved performance is that the reinforcement algorithms adapt to the traffic by being less bound to only one path per destination than the case for BF. The percentage of which peers should forward which messages to a particular receiver is stored in a proportion vector $\vec{p}$ ($\Sigma_i p_i = 1$, if the BF algorithm had such a vector it would store a 1 in precisely one of the positions in the vector)

These promising results by the applying machine learning for efficient routing motivates the support for pluggable routing algorithms in the architecture, as shown in figure 6.

2.9 Resolver Agent Architecture

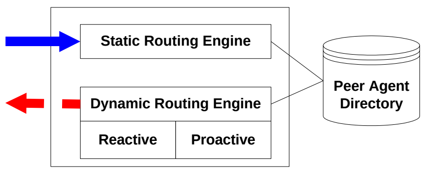

Figure 6: Resolver Agent Architecture

Static Routing Engine supports simple routing based on routing table lookups.

Dynamic Routing Engine supports two types of routing. The first is the *Reactive* router, it routes or prunes the current incoming message (i.e. stimulus-response on incoming messages).

The second is the *Proactive* router, it tries do larger changes such as altering topology based on clustering-ideas in excess cpu periods. The motivation behind this is to try to reduce the number of network hops for ACL messages by putting semantically similar agents closer. In the case of a recommender system, this could be done by calculating the proximity between every pair of peers of resolver agents (based on historic messages to/from the peer) and notify all pair of peers that has a high resemblance [16].

Peer Agent Directory is the routing table, with information about other jfipa nodes and agents.

Other tasks of the proactive router could be to try to estimate changes in topology based on analysis of messages (frequency, resolver information, content etc.), in order to optimize the routing tables. Interaction detection is another possible task, e.g. figuring out when there is an auction and which type of auction etc. The proactive router could also do statistical calculations and estimates in order to try to improve performance.

Interactions with other resolver agents related to feedback on routing decisions is either done using the inbox/outbox FIPA-ACL based network IO mechanisms (possibly with the development of routing ontologies), or by proprietary network IO in the Dynamic Routing Engine.

3 Implementation

The implementation of jfipa is still work in progress, so far a set of efficient parsers (for HTTP, XML-encoded FIPA Envelopes and Messages) have been made, as well as the overall architectural choices.

3.1 jfipa benchmark

Figure 7 compares jfipa parsing performance with state-of-the art XML parsers. Crimson is Sun Microsystems own XML parser, now taken over by the Apache project, and Xerces is being described as “the next generation, high-performance XML parser”. Both were clearly outperformed by jfipa[2].

[2]Measurements were done on a Dell Inspiron 4100 w/1GHz PIII CPU, 512MB ram, W2K

Each measurement point is the average of 10000 parsing rounds of a FIPA XML Envelope, this is done in order to make the parsers initialization costs neglible. A spreedsheet containing measurements and test-files can be found at jfipa.org/publications/2002/jfipaBenchmark.xls.

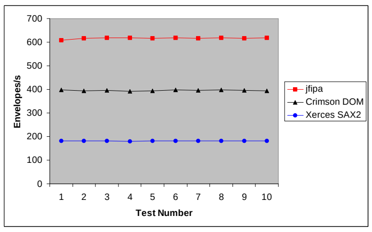

Figure 7: jfipa XML Parsing performance

One of the main reasons for the performance differences performance is that Crimson and Xerces need to build an memory model at runtime of the XML DTD in order to validate the XML document (XML-encoded FIPA Envelope in the test case). jfipa allready has this model pre-runtime since it is specialized towards handling XML-encoded FIPA ACL Envelopes and Messages. Other optimization approaches in jfipa is efficient reuse of allocated objects, avoidance of slow String-based operations, and buffered IO.

3.2 Code structure

The code structure of jfipa is divided in three main packages:

1. api
2. implementation
3. test

The test package is testing the api methods (i.e. interfaces), but with the selected implementation. The motivation behind this structure is that it should be easy to test new implementations with existing test code, e.g. porting the implementation package to support java for mobile devices (J2ME).

3.3 Code quality

In order to ensure that the jfipa code is fairly solid, unit tests are created, this allows quick retesting of the *whole* system each time changes are made.

3.4 Open Source Licence

jfipa is under the MIT Open Source licence. As opposed to the Gnu Public Licence (GPL) and the

Lesser Gnu Public Licence (LGPL), the MIT open source licence is rather non-problematic regarding commercial utilization of the software.

3.5 jfipa Code Availability

The jfipa source code is being hosted by VA Software's SourceForge.net on http://jfipa.sf.net/. For documentation and examples, the jfipa homepage - http://www.jfipa.org is the place to look.

4 Related Work

The architectures that resemble jfipa most, is JATLite [7] and JATLite ACL. JATLite supports messages of the Knowledge Query and Manipulation Language (KQML), and JATLite the messages with the Lisp-syntax of FIPA ACL. Lately there seems to have been little development on JATLite, and the code doesn't take advantage of newer java language opportunities.

The SoFAR multi-agent framework [4] is designed for distributed information management tasks in a grid environment, it operates on a higher abstraction level than the message-oriented JATLite and jfipa, since it is centered around agent services and is geared towards human users and not only agents.

There also exists several other FIPA-based multi-agent frameworks (e.g. FIPA-OS [10] and Jade [1]), but they are much more comprehensive since they seem to try to support the *whole* FIPA standard, and not as geared towards routing and brokering mechanisms as jfipa.

5 Conclusion

In this paper an architecture for agent-based grid computing has been presented. The main contribution is the architecture itself, as well the (in-progress) open source implementation of it.

The jfipa architecture is pluggable with respect to routing algorithms, its API is evolving towards becoming simple to use (the current jfipa implementation is a refactoring of the first implementation with emphasis on simplifying API), the (parsing) processing performance is high, and the implementation is fairly robust mainly due to the many unit tests applied.

Ongoing and future work include finishing the implemention the architecture and apply it in large-scale grid experiments of data mining in massive multiplayer computer games. The primary research problem will be knowledge discovery of player logoff patterns in order to get 1) a player stability metric for massive multiplayer games, and 2) being able to do counter-actions, e.g. automatic recommendations based on collaborative filtering in order to keep players longer (and generating more revenue for the game provider). Integration of jfipa software with Agent-Oriented Software Engineering methodologies is also a possible direction, we refer to [15] for a overview of such methodologies.

Other areas considered investigating is the simulation of financial products (e.g. derivatives) related to the upcoming market of carbon dioxide quoate trading (see PointCarbon.com). jfipa will eventually be integrated with the Agora Multi-Agent framework [8], in order to make Agora support FIPA ACL communication.

Acknowledgements

I would like to thank my Agentus colleague Thomas Brox Røst for fruitful discussions on java performance, as well as my supervisor Mihhail Matskin. This work is supported by the Norwegian Research Council in the framework of the Distributed Information Technology Systems (DITS) program and the (ElComAg) project.

References

[1] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Jade - a fipa-compliant agent framework. In *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM)*, pages 97–108, 2000.

[2] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[3] Chris J. Merz David H. Wolpert, Sergey Kirshner and Kagan Turner. Adaptivity in agent-based routing for data networks. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 396–403. ACM, june 2000.

[4] Luc Moreau et al. SoFAR with DIM Agents. In Jeffrey Bradshaw and Geoff Arnold, editors, *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*, pages 369–388, Manchester, UK, 2000. The Practical Application Company Ltd.

[5] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organization. *The International Journal of High Performance Computing Applications*, 15(3):200–222, Fall 2001.

[6] Erich Gamma, Richard Helm, Ralph Johson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[7] Heechol Jeon, Charles Petrie, and Mark R. Cutkosky. Jatlite: A java agent infrastructure with message routing. *IEEE Internet Computing*, pages 87–96, March–April 2000.

[8] Mihhail Matskin, Ole Jørgen Kirkeluten, Svein Bjarte Krossnes, and Øystein Sæle. Agora: An infrastructure for cooperative work support in multi-agent systems. In Tom Wagner and Omer F. Rana, editors, *Infrastructure for Agents, Multi-Agents and Scalable Multi-Agent Systems, Lecture Notes in Computer Science*, volume 1887, pages 28–40. Springer-Verlag, 2000.

[9] Bernhard Moulin and Brahim Chaib Draa. An overview of distributed artificial intelligence. In Greg M. P. O'Hare and Nicholas R. Jennings, editors, *Fundamentals of Distributed Artificial Intelligence*, pages 3–56. John Wiley and Sons, 1996.

[10] Stephan Poslad, Phil Buckle, and Robert Hadingham. Open source, standards and scalable agencies. In *Proceedings of the fourth International Conference on Intelligent Agents.* ACM, 2000.

[11] Jon Postel and Joe Touch. Network infrastructure. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 21, pages 552–562. Morgan Kaufmann Publishers, Inc., 1999.

[12] Omer F. Rana and Kate Stout. What is scalability in multi-agent systems? In *Proceedings of the fourth International Conference on Intelligent Agents*, pages 56–63. ACM, june 2000.

[13] John R. Searle. *Speech acts: An Essay in the Philosophy of Language.* Cambridge University Press, 1969.

[14] Gerard Tel. *Introduction to Distributed Algorithms*, chapter 4, pages 103–154. Cambridge University Press, 2 edition, 2000.

[15] Amund Tveit. A Survey of Agent-Oriented Software Engineering. Proceedings of the First NTNU CSGS Conference, http://www.jfipa.org/publications/AOSE/, May 2001.

[16] Amund Tveit. Peer-to-peer based recommendations for mobile commerce. In *Proceedings of the First International Workshop on Mobile Commerce*, pages 26–29, Rome, Italy, July 2001. ACM.

[17] Michael J. Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 2(10):115–152, 1995.