The Joint Symposium
ASA/MA2000

ETH Eidgenössische
Technische Hochschule
Zürich

Poster Abstracts

Contents

Thursday, September 14
16:45 - 18:15

Mobile Agent: Enriching Document Management and Distribution for Mobile Design Work

Mark Allen Dr. Geof Staniford Professor A. Taleb-Bendiab
M.Allen@livjm.ac.uk G.Staniforn@livjm.ac.uk A.Talebbendiab@livjm.ac.uk

Abstract and Structure

This poster describes work in progress which is focusing on the application of multi-agent systems to support distributed collaborative work in a large scale industrial setting (the Enrich project). The poster provides a brief description of the industrial problem, and then presents an argument factorisation technique, which is used to support argument generation for the overall purpose of reducing information overload to system designers. In addition the poster details our proposed systems architecture as well as work in progress upon the theoretical basis for our design and implementation.

Background

The Enrich project is concerned with supporting distributed design through the use of resource management techniques. Under development is an Inventory Management System to facilitate product development for design engineers. To help them access and manage design resources and share information and data with other partners.

The primary aim is to provide an inventory of documents and other resources related to a project. Each user will be able to view the inventory in a variety of ways that suit them best. They will be able to add documents, collaborate with other users, change documents and view documents from their own and other projects. No guarantee can be made as to the type of the document and the necessary applications will also be stored in the information repository.

Users will need to be informed of any changes to the documents they own, or are currently tracking. They will also need an automated, and transparent, method of tracking new documents within subject areas they have registered an interest in.

The shared information repository will be highly distributed on a global scale. The server side of the system is to be built using Suns' Java Enterprise Edition (JEE), which will be extended by a multi-agent systems layer.

Argumentation-based system

It is likely that vast amounts of data will be available to the designer. The problems of information overload have been highlighted as an important issue in the world of distributed systems and the World Wide Web. It is therefore necessary to add to the Enrich Inventory Management System methods of filtering the available data.

There has been much research into information filtering in recent years. Many of these rely on keyword searches, which have proven unsuccessful in very large databases. Ideally natural language processing would be used but this is some way from practical implementation in large distributed systems.

In our system the document repository inventory is defined by a tree of XML documents that describe content, location, necessary software and other meta information about documents. This tree will become very large indeed and navigation, by designers, through the virtual space is an extremely important issue. An argument generator previously developed in the domain of artificial intelligence and law is to be adopted to support collaborative conflict resolution and design resources management activities. This is a case based multi-agent system that uses a compositional mechanism to construct, from previously factorised cases, a rhetorical argument in favour of a given legal case.

The factors of a case are extracted from the original document through a knowledge elicitation process. Each pertinent fact becomes a factor that tends to support one side or the other of the rhetorical argument. Factors can be binary or some enumerated value. External information relevant to a case is dealt with through the concept of abstract factors. These form a multi-level hierarchy in which atomic factors support or militate against higher level abstract factors.

It is envisaged that this system can be developed to argue that a particular document should be included in a users view of the inventory. The users preferences are known to the system, subject area, roles and responsibility. The user can also declare interests in particular subjects. We intend to specify this type of information to the agents in a (normative) deontic specification that is based upon a computable form of first order predicate logic. Armed with such information an intelligent and autonomous agent roams the distributed inventory accessing documents and building an argument for the inclusion of the document in the users list. If the argument is strong it will be passed on to the user for him to make his own decision. If it is not (the argument system produces both sides of the debate) the document is dropped and the agent moves on to the next one.

The factorisation of the documents will be based on the XML schema, which are attached to all documents, groups of documents, subjects and projects in the information repository. This factorisation is a development from the legal case system. The fact of a case now replaced by the semantic meaning of a section of text.

Why Mobility?

The system is designed for large data sets. These will be distributed over many servers. We consider that moving many, possibly large, documents over the network is inefficient and unnecessary. Only the agent (or resource manager) and its meta information including arguments needs to be transmitted. The document itself will only be transmitted if the user is persuaded by the agent's argument and specifically requests it. It would therefore be more suitable for the agent to travel to the data store and carry out its work at the site. It would then return with any arguments considered strong enough and present them for the user to review.

A second, important, use for mobile agents in the Enrich environment is to support maintenance, both of the system itself, and of the inventory. For example if a new document is added to the inventory other documents will need their XML file updated to hold a reference to the new document. As some of these documents are likely to be on another server somewhere we will create an agent to propagate the new information through the system.

Future Work

There are two areas of research that need to be undertaken. One we need to experiment with the argument generator to determine the boundaries for which readable arguments can be created, and to show empirically that they are useful – that is the users number of wrong "hits" is significantly reduced. We also need to investigate if it is possible to automate the generation of the factors of a document. We envisage this part of the system as an application of natural language processing supported by machine learning. Initially the factorisation would be poor but with feedback from the human user the activity would become progressively refined.

This poster will describe a work in progress

Fault-Tolerant Mobile Agents in Mozart

Iliès Alouini and Peter Van Roy*

Introduction

In any wide-area distributed system such as the Internet, fault tolerance is crucial for real-world applications. We describe a new practical fault-tolerant mobile agent platform. The agent platform is built on the top of a global store abstraction [1, 2] that provides a globally coherent and fault tolerant memory. The global store abstraction is implemented as a user library that runs on the Mozart platform. The implementation does not require recompiling the platform. Mozart is a general-purpose development platform for open, robust distributed applications that is based on the Oz language [3]. The global store implementation is based on Mozart's reflective fault model and takes full advantage of the platform's network-transparent properties.

The global store abstraction

A *global store* consists of a set of objects replicated on several processes. A *user* is any computation that is part of an OS process running Mozart. Users can connect to or disconnect from a store dynamically. This adds or removes processes from the global store. A user invokes objects by initiating a transaction, which calls the objects. It is possible to use the store so that a user's object updates are seen instantaneously by that user without waiting for the network. This implies a possible speculative execution, which is completely managed by the store. Users invoke objects without worrying about concurrency control or store failure. Both concerns are managed by the store. Because Mozart is network transparent, users can communicate and collaborate by *sharing* a global store. The global store tolerates any number of user failures, as long as it exists on at least one process. The store can migrate without dependencies, i.e., the migration depends on no fixed process.

The store is lightweight and requires no persistence to recover from failure. The store uses process redundancy; with n processes it tolerates up to $n - 1$ fail-stop process failures.

Mobile agents using the global store abstraction

One of the challenges for mobile agent platforms is to provide robust and fault-tolerant mobile agents [4, 5]. We build an agent API on top of the global store that provides fault tolerance, agent mobility without site dependencies, and permits home communication without any dependencies. In general, an agent can create any number of global stores. A first global store is used for the agent's own state, so that it can migrate. A second global store is shared between agents, for communication. The second global store is used to implement Send & Receive operations in a few lines of code. Our agents have the following properties:

- An agent can move from one site (i.e., OS process) to another, e.g., to reduce network latency.
- The agent's internal state is maintained when moving.
- Agents live in Mozart's shared computation space and can therefore communicate any data including compound structures, procedures, classes, etc.

*Université catholique de Louvain, Département d'Ingénierie Informatique, B-1348 Louvain-la-Neuve, Belgium. E-mail: {ila,pvr}@info.ucl.ac.be

- An agent can continue to function despite network inactivity (disconnected operation).
- Agents are not affected by process or host failures, if at least one process survives somewhere.
- Agents do not reference the file system when they move (except possibly initially, when creating a global store). When an agent moves from one site to another, there is no need to load agent classes from a file system, e.g., like in IBM Aglets. The agent classes are transferred in a transparent way through interprocess communication.

Mobile agent API

We define a *mobile agent* to be any distributed computation that has the ability to move from one site to another. An *agent* is a set of concurrent tasks where a *task* is a computation that uses the resources of a single process and can communicate or collaborate with other agents. Here is an API that allows to program an agent:

- Agent creation: `MA={New Agent.agent init(NewObj AgentStore)}`, with arguments: `Agent.agent` (input agent class), `NewObj` (returned procedure to create new objects in the agent store), and `AgentStore` (returned reference to agent store).
- Execute the task `F` of agent `MA`: `{MA run(F)}`, with argument `F` (a task). In Mozart, the task is defined as a *functor*, which is a first-class data structure defining a component specification. A functor defines the process-specific resources the task needs. The functor `F` can be created on the fly during task execution.
- Move the agent `MA` to host `IPadd` and execute `F` remotely: `{MA move(IPadd F)}`, with arguments `IPadd` (IP host address) and `F` (a task). The original task is not moved, but because the global store contains the agent state and depends on no fixed process, the result is a move whose strength is intermediate between weak and strong mobility.
- Move to home site: `{MA movehome()}`.
- Communicate with other agents: `{MA send(M)}` (send asynchronously message `M` to agent `MA`) and `{MA receive(M)}` (receive message `M` at agent `MA`).

Conclusion

This extended abstract explains the highlights of our simple mobile agent platform. We have shown how a fault-tolerant agent platform can be built naturally using a general-purpose platform for open distributed computing augmented with the global store abstraction. Our current research includes building other high-level abstractions for fault tolerance, secure distributed programming, and the implementation of agent-based applications.

References

[1] Iliès Alouini. *Global Store Module.* Available at `http://www.mozart-oz.org/mogul/info/alouini/globalstore.html`, April 2000.

[2] Iliès Alouini and Peter Van Roy. *A Practical Fault-tolerant Store Abstraction for Multiple Application Domains.* To be submitted, July 2000.

[3] Mozart Consortium. *The Mozart Programming System (Oz 3).* Available at `http://www.mozart-oz.org`, January 1999.

[4] Holger Pals, Stefan Petri, and Claus Grewe. *FANTOMAS: Fault Tolerance for Mobile Agents in Clusters.* IPDS 2000 Workshops, LNCS 1800, pages 1236-1247, 2000.

[5] Detlef Schoder and Torsten Eymann. *The Real Challenges of Mobile Agents.* Communications of the ACM (43) 6, pages 111-112, June 2000.

A Knowledge-based Internet Agent System with a Formal Verification Facility

Tadashi Araragi

NTT Communication Science Laboratories
2-4 Hikaridai, Seika-cho, Souraku-gun, Kyoto, 619-0237 Japan
e-mail: araragi@cslab.kecl.ntt.co.jp
http://www.kecl.ntt.co.jp/csl/ccrg/members/araragi

Overview

In this poster, we present a knowledge-based agent system focused on Internet applications like e-commerce. This system has a logic-based programming language and a formal verification facility for agent programs written in this language. With this language, we can express, in a simple way, the characteristic behaviors of Internet agents, such as agent name passing, dynamic retrieving and loading of programs, and migration. In addition, we can verify its agent programs based on CTL model checking. This system is implemented in Java, and we call it Erdös.

Programming of Erdös

One of the characteristic features of Erdös's programming is division of agent communication protocols and the procedures that do not require communication between agents. Erdös's programming language is used to describe the protocols, and the procedures that do not need agent communication are implemented in Java as external methods that are called from Erdös's agent program. The language stems from Halpern's knowledge-based program. Each agent of Erdös has a knowledge base that consists of logical formulas. The program consists of subprograms, and each subprogram is a sequence of "test-actions" that are executed in the order. A test-action has the form "If *test* then $action_1$, ..., $action_m$ else $action_{m+1}$, ..., $action_n$;" The *test* is a logical formula. In execution of this test-action, if the test formula is derived from the current knowledge base, then $action_1$, ... are executed, otherwise $action_{m+1}$, ... are executed. For this derivation, we support a restricted first-order logic and a logic of belief (KD4). We have selected a few actions for Internet agents: *add*, *rm*, *call*, *idle*, *ex_call*, *go*, *create*, *stop*, *resume*, *kill*. Example fragments of a program are given below.

```
if Info(ds_name,?ds) and Req(ds,?msg)
   then add(?ds: Req(self,?msg)), rm(Req(ds,?msg)); --(1)
if Info(ds_name,?ds) and Info(?ds,?msg)
   then call(self: contract_subprg) else idle; --(2)
if Info(cond,?cond) then ex_call(judge-condition,Arg(?cond));  --(3)
```

In (1), "?ds" is a variable that is instantiated in the test process, and the substitution is done over the test-action. "self" is also a variable instantiated by the name of the agent executing the test-action. In this fragment, the agent checks if it has the name of a directory sever and a message to send to the sever. If it finds these, then the message (the formula Req(self,?msg)) is added to the knowledge base of the directory server (i.e. the message is sent). Here, note that a correspondent is dynamically determined by using a variable(?ds). In (2), if the test is successful, the agent calls the subprogram contract_subprg, otherwise it waits in the idle action until the test becomes successful. Using the idle action, agents can synchronize their behaviors. If an agent name is specified in the call action instead of “self”, the subprogram is dynamically downloaded from the agent. In (3), the agent judges whether the condition is acceptable by using the external method judge-condition through the ex_call action. The return value is given in its knowledge base in a logical formula.

Verification of Erdös

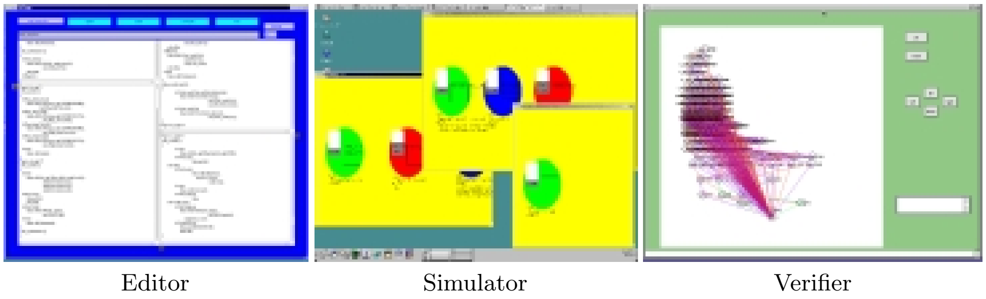
Agent programming is a type of distributed programming. Therefore, the verification of program is a difficult task because of its asynchronous behavior. Erdös offers a formal verification facility based on the well known model checking algorithm of CTL. The essential point of our verification method is the transformation of our agent programs, which involves a deduction procedure on the knowledge base, to a Boolean formula of the total transition relation of the asynchronous behaviors of agents. CTL model checking deals with only finite state systems. In Erdös's verification, if the programmed system is infinite state, we abstract the infinite state parts of the program manually so that the system becomes finite state. The simple structure of Erdös's programs as well as the isolation of external methods enables this verification and abstraction. For many agent systems that consist of Java and agent libraries, it seems impossible to formally verify their programs directly.

Programming Environment of Erdös

Our programming environment is illustrated in Fig. 1. The left side is an editor with a function of checking the syntax of agent programs. The center is a simple visual simulator. Each rectangle expresses a host machine, and the ellipsoids in the rectangles express agents. For each agent, we can monitor the state of the execution, the knowledge base, and the currently executed test-action. We can change the processing speed of each rectangle manually. The right side of the figure is our verification tool. CTL model checking is calculated with BDD, and its performance is very sensitive to “variable order”. Our tool enables us not only to verify CTL specifications but also to observe the structure of BDD during verification so that we can analyze the proper variable order.

References

1. T. Araragi and K. Kogure: *Dynamic Downloading of Communications Protocols Using a Logic-Based Agent System*, Proc. of Workshop on Computational Logic in Multi-Agent Systems (CLIMA-00), Imperial College, London, pp. 27 - 34, 2000.
2. T. Araragi, P. Attie, I. Keidar, K. Kogure, V. Luchangco, N. Lynch and K. Mano: *On Formal Modeling of Agent Computations*, Proc. of the 1st Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS 2000), LNCS, Springer, 2000 to appear.



Editor

Simulator

Verifier

Fig. 1. Programming environment

Active Networking, QoS and Virtual Routers

Florian Baumgartner[†] and Torsten Braun
Institute of Computer Science and Applied Mathematics
University of Berne, Neubrückstrasse 10, CH-3012 Berne

Introduction

The points of interest for the mapping of resource reservations are the borders of regions. Different regions may support different reservation protocols. This is not necessarily the entire network of an ISP. Even within an ISP it might make sense to provide several areas with different supported reservation methods. Because of this we will not refer to an ISP's network but to a so called Reservation Domain (RD), representing a certain topology regarding reservation protocols or administration.

At the border of a RD several mappings may be necessary. Obviously the RSVP to DiffServ mapping is the most probable case [BBBG00]. Because of scalability issues an ISP will be interested in mapping RSVP reservation to Differentiated Services decreasing the load of his backbone routers. On the other hand even the mapping of different Differentiated Service (DS) classes may be necessary, as it is finally left to the an ISP, which Differentiated Service Code Points (DSCP) are used for which type of traffic.

Mobile Agents and Service Mapping

Because of the dynamic behaviour of such RDs and the overhead of a central instance, an approach of using Mobile Code to translate the different resource reservation schemes is favoured. After the injection of capsules, agents occupy the borders of a homogeneous reservation domain, supervising incoming reservations. In the case of DS mapping the agent will determine an appropriate mapping for the packet, reconfigure the border router to do the proper mapping and traffic conditioning and forwards a capsule along the packet's path through the reservation domain in order to configure appropriate scheduling mechanisms in each router.

A more complex task than the support of different DSCPs is the RSVP-based QoS support. RSVP is based on an end to end scheme, so the RSVP messages used for the setup of an reservation have to be transported through the RD. We propose an approach to dynamically negotiate tunnels between the ingress and egress points of an RD as well as the setup of tunnels spanning multiple RDs (see figure 1 left) requiring an interaction of different agents. The tunnels are setup using mobile code to establish the tunnel endpoints.

Beneath the general flexibilty of the approach using active elements, this has the advantage, that a capsule being sent by a border router to the final destination of the data stream passes automatically those routers, which have to be configured. So no knowledge about the the network within an RD is necessary, while a more central approach would require some kind of topology database.

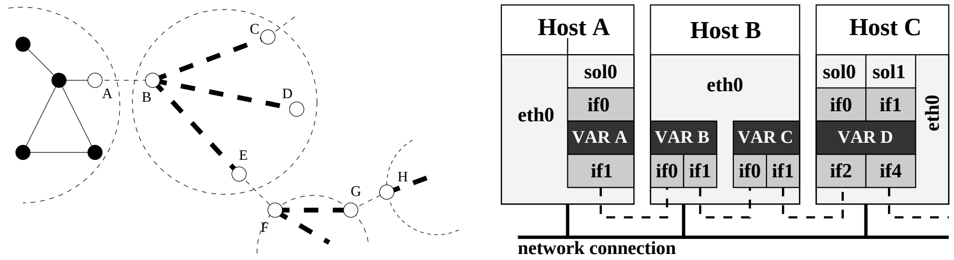


Figure 1: left: Reservation Domains with Agents located at the RD borders and tunnels setup by Agents at ingress and egress points; right: multiple hosts running Virtual Active Routers for emulation purposes

Concept Evaluation by Virtual Active Routers

To evaluate these concepts, to encounter the problem of combining IP forwarding technologies and Active Networking and to be able to provide a testbed of sufficient size we developed an approach to emulate complete active routers including the appropriate IP forwarding [BB00a]. This allows the combination of real hardware and routers with large emulated topologies using several so called Virtual Active Routers (VAR) running on the same host. This basic idea of combining real hardware with an emulated topology is shown on figure 1 (right). The concept simplifies the setup of big topologies on only an couple of real hosts and allows to integrate real hosts and applications with these emulated topologies. A real host cannot distinguish between real and emulated network. Being equipped with capable queueing systems [BB00b] and routing mechanisms the VARs provide a great platform for the easy implementation of new components and concepts. A Capsule Interpreter running on each VAR is used to evaluate the resource reservation mappings.

References

[BB00a] Florian Baumgartner and Torsten Braun. Quality of service and active networking on virtual router topologies. *The Second International Working Conference on Active Networks*, October 2000.

[BB00b] Florian Baumgartner and Torsten Braun. Virtual routers: A novel approach for qos performance evaluation. *Quality of future Internet services, QofIS'2000*, September 2000.

[BBBG00] Roland Balmer, Florian Baumgartner, Torsten Braun, and Manuel Günter. A concept for rsvp over diffserv. *IEEE, ICCCN'2000*, October 2000.

Experiences with State-of-the-Art Migration Strategies

Peter Braun, Christian Erfurth, and Wilhelm Rossak

Computer Science Department
Friedrich Schiller University Jena
D-07740 Jena, Germany
Peter.Braun@informatik.uni-jena.de
http://tracy.informatik.uni-jena.de

1 Introduction

In the last years, research and development of mobile agents made a great leap forward. Along with the wide spread of Java based applications, mobile agents became extensively popular not only in research, but also in industrial projects. Research in the area of mobile agents is looking at languages that are suitable for mobile agent programming [2], and languages for agent communication [1]. Very much effort is put into security issues [3]. Several prototypes of real-world applications in the area of information retrieval, management of distributed systems, and mobile computing are in development.

The performance aspect of Java-based mobile agents has not been considered in literature, so far. In our opinion, performance is of increasing importance as mobile agents are disseminated in more and increasingly diverse application areas. Within the life-cycle of a typical mobile agent, we find several occasions at which performance can be improved. We can divide these occasions into two classes according to *transmission aspects* and *runtime aspects*. In the first class, all techniques are summarized that influence network load and transmission time during agent migration. Transmission time is influenced by network bandwidth and network latency, and can be reduced by code compression techniques and by restricting the number of class files and data items that must be transmitted. In the second class we place techniques by which an agent's execution time can be improved, e.g. by using a sophisticated Java Virtual Machine.

In this paper we only deal with transmission aspects. Especially, we are interested in performance effects that result from the *migration strategy*, i.e. the way how code and data are transmitted during migrations. An agent typically consists of several class files and a lot of data and state information. The *push-all-to-next* strategy transmits the agent's complete code and the serialized agent (i. e. data and state), as one package to the next destination platform. In contrast, the *push-all-to-all* strategy transmits the complete code to *all* platforms the agent will migrate to, but the agent's data and state information only to the first platform in the given itinerary. The *pull-per-unit* strategy transmits only data and state to the next platform. Code must be downloaded on a per-class policy on demand. At last, in the *pull-all-units* strategy, all classes are downloaded as one archive at once. We will show results of a performance evaluation of these migration strategies. It will become clear that the migration strategy can influence performance in a non-neglectable way. First experiments indicate that dynamic class loading and code size influence the agent's performance.

2 Experimental Results

We show first results of experiments that we are performing using our mobile agent system *Tracy*, a general purpose mobile agent system, implemented on top of the Java 2 platform. The main difference of Tracy, as compared to other mobile agent systems, is that it provides a migration model that offers a flexible alternative to a pure agent model and a pure network transmission model. From the agent's view it means that the agent server offers a multitude of different migration strategies the agent can choose from. From the researcher's point of view this means that the complete process of migrating an agent is accessible to the programmer and can be adapted, so that new migration techniques can be implemented.

To perform the experiment we chose an application from the information retrieval domain. Each platform has a database with documents of different types. Each document is characterized by a set of keywords. The agent has to visit each platform. First, it filters all documents according to a given set of keywords. The result is a set of *interesting* documents. Second, all these documents are examined in detail, which results in the set of all *significant* documents from which the agent takes a copy before migrating to the next platform. To examine an interesting document, a specific class file for the given document type is necessary on the current platform. Therefore, an agent consists of one class file for the agent itself, which contains code to perform the first step and all auxiliary tasks Additionally, there are five other class files, each for one document type, which contain special code for the second step. If the agent finds a document of a specific type, the corresponding class file must be downloaded dynamically.

The experimental setup consists of a cluster of five agent systems connected via a local area network which can be classified as a homogeneous network. On each platform we can change the number of document types that the agent will find interesting. By this we can directly influence the number of classes that will be downloaded. In the first experiment, the agent class files are very small. The main class is about 12 kByte, and each of the additional class files is about 2 kByte. As a result, it can be seen that the execution time has an upward trend for all migration strategies with the number of interesting document types. This is because the agent's data increases as more interesting and significant documents are found. As could be expected, strategies push-all-to-next, push-all-to-all, and pull-all-units are almost equal in time. In a case where no interesting documents exist, the pull-per-unit strategy is faster than all other methods because only the agent class itself must be transmitted. However, even if only one additional class file must be loaded strategy pull-per-unit is about 17% slower than the push-all-to-next strategy. With increasing number of document types the pull-per-unit strategy is in average more than 23% slower than the push-all-to-next strategy. This performance difference only results from the fact that code must be downloaded dynamically.

In the second experiment we inflated class files to lengthen transmission times. The agent class file is about 12 kByte, again, the additional class files are about 25 kByte, now. If no or only few additional class files must be downloaded, pull-per-unit strategy is now up to 33% faster than the other strategies. If more than two additional class files must be downloaded, this strategy is slower than all other strategies for the same reason mentioned above. In the case that all classes must be downloaded, this strategy is about 12% slower as compared to the other strategies. With longer class files, the pull-per-unit strategy performs better for more document types because of the overall difference of the network load.

3 Conclusions

In this paper we gave an brief overview of state-of-the-art migration strategies and compared them with regard to transmission time. It could be seen that significantly different migration strategies may lead to almost the same execution time. From these experiments can also be concluded that the pull-per-unit strategy, which is used in several mobile agent systems today, is a bad choice in some cases. Notwithstanding, we were able to show that there is no migration strategy that is best in every situation. Additionally, on basis of our results, it can be inferred that a mobile agents' performance depends on several factors, and not at least the dynamic behavior of the agent which influences class file downloading. Most of these parameters are inherent dynamical, i.e. they can not be predicted in advance. In conclusion we may say that our results indicate that it is worthwhile to decide which migration strategy a mobile agent should use. However, this decision process must be based on parameters that can not be known by the programmer in advance.

References

1. J. Baumann, F. Hohl, N. Radouniklis, K. Rothermel, and M. Straßer. Communication concepts for mobile agent systems. In K. Rothermel, editor, *Proceedings of the First International Workshop on Mobile Agents (MA'97), Berlin, April 1997*, volume 1219 of *Lecture Notes in Computer Science*, pages 123–135, Berlin, 1997. Springer Verlag.
2. G. Cugola, C. Ghezzi, G. P. Picco, and G. Vigna. Analyzing mobile code languages. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems: Towards the Programmable Internet (MOS'96), Linz, July 1996*, volume 1222 of *Lecture Notes in Computer Science*, pages 93–110, Berlin, 1997. Springer Verlag.
3. G. Vigna. *Mobile Agents and Securtiy*, volume 1419 of *Lecture Notes in Computer Science.* Springer Verlag, New York, 1998.

MobiliTools: A Toolbox for Agent Mobility and Interoperability Based on OMG Standards

Bruno Dillenseger
France Télécom R&D (ex-Cnet), BP98, F-38243 Meylan Cedex, France
bruno.dillenseger@rd.francetelecom.fr

Introduction to MobiliTools

Mobile agent platforms typically come with a fixed combination of a communication model, an activity model and a mobility model. As a matter of fact, mixing autonomous activities with remote communication and mobility requires an accurate assembly job in practice.

But is this a sufficient reason for turning mobile agent platforms into strongly integrated frameworks? Such an approach leads to specific platforms that are dedicated to specific needs, while there is no ultimate agent model suited to every need. Moreover, these heterogeneous specific platforms typically don't interoperate very well.

However, several standardization efforts are in progress in the agent technology field (e.g. FIPA, OMG's Mobile Agent System Interoperability Facilities). Voyager's CORBA support and Grasshopper's MASIF and FIPA compliance show encouraging efforts towards interoperability.

In this context, MobiliTools intends to provide a set of "middleware" components that make it easy to build a number of customized, ad hoc mobile object/agent platforms, with various communication and activity models, while providing a basic level of interoperability support, based on OMG standards (CORBA, MASIF).

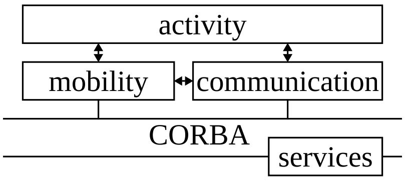


Fig. 1. MobiliTools architecture

These components can be used either together, or independently in other mobile agent platforms, to introduce interoperability bridges.

SMI, Simple MASIF Implementation

SMI is a minimal and generic MASIF implementation playing the role of MobiliTools' mobility component. MASIF's framework consists of *mobile agents*, identified by a unique *name*, acting on behalf of an *authority*, moving from *agent system* to agent system, and executing in *places*. Agent systems also have unique names, and are bound to an authority. Agent systems implement the MAFAgentSystem CORBA interface, mainly for managing agents lifecycle and mobility. A directory and lookup service for agents, places and agent systems, is also available through the implementation of the MAFFinder CORBA interface.

SMI straightforwardly implements this framework in Java, while adding a couple of operations that are lacking in both MAFAgentSystem and MAFFinder interfaces. SMI provides two possible levels of customization, through the open implementation of two Java interfaces:

1. interface MobileObject has to be implemented by agents in order to react to lifecycle events: creation, successful or aborted mobility, activity suspension or resumption, death, and agent system shutdown. These events trigger dedicated agent call-backs, which have to be defined by the programmer, to actually manage agent activity;
2. interface AgencyPersonality may be optionally implemented by the programmer in order to prepare for, and then acknowledge, agent lifecycle events. An agency personality may also wrap every agent in another MobileObject implementation in order to trap lifecycle events and serialization. As a result, an agency personality makes it easier to implement a management kernel or scheduler for agent activities, while customizing an SMI generic agent system into a specific agent system.

At the present time, a passive agent model and an active agent model with one thread per agent have been (very quickly and easily) implemented. Another model, based on a synchronous programming model, is under construction and promising, especially regarding scalability and transparency issues. We are also envisaging the implementation of a sort of ODP “cluster” model.

ACTS, Agent Communication Transport Service

ACTS is a CORBA service for transporting messages between heterogeneous agents, whether mobile or not, and whether CORBA objects or not. It can be operated in both a message push and a message pull model. It may be wrapped and customized in a number of ways, to implement a variety of communication models and architectures, but still supporting interoperability at transport level. For instance, private mailboxes with name-based addresses, multicast and unicast features, and a FIPA-compliant communication architecture have been implemented.

Although it is independent of MASIF, ACTS may be regarded as a complement enabling interoperability between agents for remote communication, through the definition of extra CORBA interfaces.

On-going work. We are developing various agent activity models, focusing on transparency, interoperability and scalability issues, for both communications and mobility.

Availability. MobiliTools is available free of charge on signed demand, for non-commercial use (mail bruno.dillenseger@rd.francetelecom.fr).

Acknowledgements. MobiliTools include contributions from Anne-Marie Tagant and Huan Tran Viet. Design discussions have involved Stefan Covaci, Alexander Yip and Jean-Bernard Stefani.

Internet Service Delivery Control with Mobile Agents

Manuel Günter and Torsten Braun
Institute of Computer Science and Applied Mathematics
Neubrückstrasse 10, CH-3012 Bern, Switzerland
http://www.iam.unibe.ch/~rvs/

Introduction

The rapid growth of the transport capacity of the Internet and the global trend towards liberalisation of the telecommunication market forces the Internet service providers (ISP) to look for new revenues beyond pure connectivity offerings. Therefore, ISPs that control their own network try to introduce new Internet services including quality features such as premium transport or traffic privacy through encryption. However, before the customer will pay for such services the following two problems need to be addressed: (1) How can the provider prove that the desired service is really delivered to the customer? For example, it is difficult to show that the provider transfers the customer's data with strong encryption. (2) For services involving collaboration of providers (e.g. end-to-end QoS) the question is how to find out who is responsible when the service quality is less than guaranteed to the customer.

It is in the interests of the customers and of honest providers that the customer is able to verify the permanent quality of a network service and to locate problems when they occur. We refer to this process as *service delivery control* (SDC).

For today's Internet services there is only very limited support for service delivery control. If a customer happens to detect a problem (which usually happens when the customer needs that service badly and does not get it), phone-calls between administrators, local measurements, and manual browsing of log-files will eventually lead to the identification of the problem source. Unfortunately, it is also not uncommon that the involved parties will suspect each other and repudiate any guilt. Note, that this problem not only concerns the relation between customer and provider but also between providers themselves. It is to be expected that the problem becomes worse when new and more expensive network services are deployed that require provider collaboration.

We propose to use a generic service delivery control architecture based on mobile agents. Mobile code allows the customer to test the service where it is delivered. Software agents as well-defined code entities facilitate the deployment of secured environments.

Mobility and Service Delivery Control

Mobile agents have been proposed for a wide range of tasks. However, code mobility has few provable advantages besides of being a catching metaphor. The following reasons describe why mobile agents are particularly useful for service delivery control agents.

- **Data source location.** Network services are per-definition delivered *in* the provider network. It thus makes sense to (at least pre-) process the measurement data there (at the source).
- **Generic interface.** By providing an agent platform at relevant sites the provider can give access in a controlled fashion.
- **Flexibility.** A general-purpose agent programming language provides the expressive power needed to cope with the unforseeable IP services of the future.
- **Trust through source code.** The customer sending the agents has insight to the agent's code. Therefore, the customer can verify what is being measured.
- **Cross checking.** A misconducting provider can easily fool a customer that relies on the measurements published by the provider. In a a multi-provider service scenario the situation is even worse. SDC agents can be sent out to perform active measurements by producing and measuring traffic at different sites. Mobility allows the agents to virtually 'track-down' the problem source.

- **Performance.** Mobile agents structure distributed computing thereby enabling the customer to collect computing power to analyse the traffic. Furthermore, mobile agents preprocess the measurement data where it is produced, thus reducing the network load.

Given these arguments we can say that even if the providers would allow their customers to access SNMP (IETF RFC 1157) agents of their equipment, the expressive power of a mobile agent based SDC approach exceeds by far what can be done by traditional SNMP based monitoring.

A Supporting Infrastructure for Service Delivery Control Agents

Agents are executed in protected node environments. We propose to locate these environments at the peering point between autonomous IP networks (see figure 1 left). The SDC architecture should not facilitate eavesdropping other customers' traffic, spoofing of foreign IP addresses or denial-of-service attacks. Given these requirements we foresee the following node architecture as shown in figure 1 (right): At the peering router, there is a *T-component* that serves as a high-performance and configurable packet copying mechanism. It adds a high-accuracy time-stamp to the packet. The T-component forwards the requested packet copies to the *Node environment*. Note, that for security reasons the agents do *not* have direct access to neither the T-component nor the packet copies. The node provides an execution environment (user-level thread in a 'sand-box') for each agent. The agent's execution environment contains an inbound and an outbound packet queue secured with a policy-based filter which ensures that the agent can only see traffic for which it is authorised.

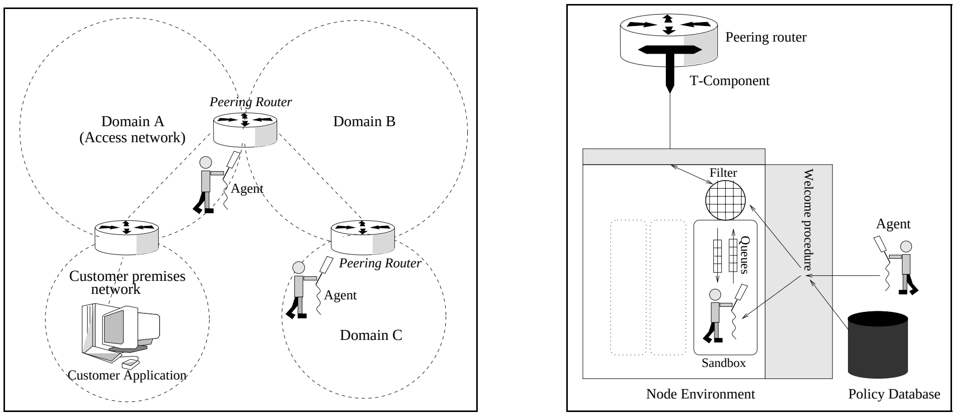


Figure 1: Measuring at peering points (left). The node environment (right).

Examples of Customer Controlled IP Services

- **IPSec (IETF RFC 2401) virtual private network control agents.** We are developing SDC agents that perform the following checks: (1) Leaking of Intranet traffic into the public Internet. (2) IPSec protocol conformity. (3) Key exchange (IKE) activity survey. (4) Statistical tests on authentication and encryption quality.

- **Differentiated services (IETF RFC 2475) control agents.** We are developing SDC agents that perform the following checks: active and passive measurement of packet loss, delay and jitter. Our goal is to develop an agent solution that can identify DiffServ providers which do not reserve the guaranteed resources.

Agent – based Virtual Laboratory

Goran Kimovski Danco Davcev Vladimir Trajkovic
Faculty of Electrical Engineering and Computer Science,
University "Sv. Kiril i Metodij", Skopje, Macedonia
gkimovski@intersoft.com.mk

1. Introduction

In this study, we try to extend our concept of Virtual Classroom [1] by adding a new service, that we call Virtual Laboratory (VL). Namely, the concept of the VL is to design a system that offers a possibility to the attendees to share different resources at once and work with them as if they were at the same place where (real) resources are. One possible example can be to control remotely a robot system in a chemical laboratory from a PC connected on Internet.

In the system design, we use agents as entities that work on different tasks in the system. In the context of the VL system, agents are seen as entities cooperating among themselves in order to accomplish a task, rather than separate entities attempting to expose some anthropomorphic behavior.

The cooperation among agents is established through the act of exchanging messages. Having in mind the flexibility and increased implementations of XML (eXtensible Markup Language) [2] as a next generation markup language on the Internet, we decided to define all of the communication protocols in the VL system using XML.

Due to the limitations on the posters size we doesn't comprehensively cover the object-oriented analysis of the system, as well as the communication protocol's XML definitions. Instead, we will present only the architecture of the VL as well as the collaboration among agents in the system

2. Virtual Laboratory Model

The VL system should represent an alternative to the "classical laboratory". The logical model of the VL system is given in Figure 1.

Every user can work with the system by using a web-based interface. The Container Agent enables different resources, with different implementations of their graphical views, to be placed on the web-based interface. In this way every specific resource can be presented with a graphical interface representing the nature of the resource and can communicate with the users in a manner that is close to the "classical".

View Agent is the key player on the client side (the web browser that the user is using to view the VL system).

The system can hold many definitions of different View Agents, since every different resource or a group of resources should present an interface of its own to the user.

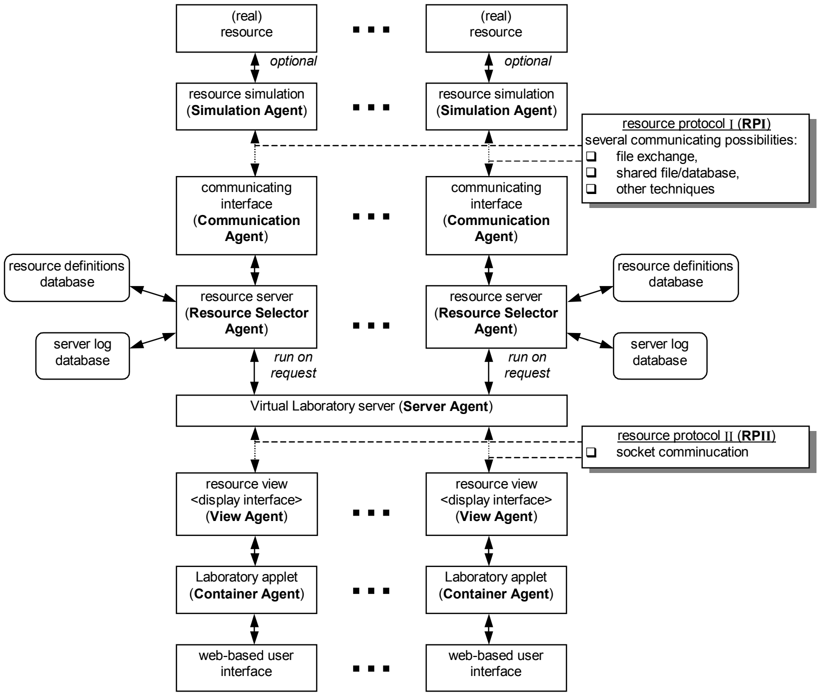

(real) resource
. . .
(real) resource
optional
optional
resource simulation (Simulation Agent)
. . .
resource simulation (Simulation Agent)
resource protocol I (RPI)
several communicating possibilities:
❑ file exchange,
❑ shared file/database,
❑ other techniques
communicating interface (Communication Agent)
. . .
communicating interface (Communication Agent)
resource definitions database
resource definitions database
resource server (Resource Selector Agent)
. . .
resource server (Resource Selector Agent)
server log database
server log database
run on request
run on request
Virtual Laboratory server (Server Agent)
resource protocol II (RPII)
❑ socket comminucation
resource view <display interface> (View Agent)
. . .
resource view <display interface> (View Agent)
Laboratory applet (Container Agent)
. . .
Laboratory applet (Container Agent)
web-based user interface
. . .
web-based user interface

Figure 1 The architecture of the VL

All of the system communication goes using two agents that run on the VL server, the Server Agent and Resource Selector Agent. Every View Agent in the system communicates with the Server Agent when a connection to the VL server is required in order to communicate with a particular resource. The Server Agent handles different connection requests and runs a Resource Selector Agent as a thread that handles the specific requests and passes back the resource responses to the View Agent.

The Resource Selector Agent's task is to get the request from the View Agent, find the appropriate Communication Agent to communicate with the specific Simulation Agent and the real resource and then pass back and forth the requests/responses between the View Agent and Communication Agent (with possible message translation).

The Communication Agent's task is to enable the message translation and communication in general with the Simulation Agent and the (real) resource. All of the agents mentioned above are to be implemented as Java classes, but the set of Simulation Agents will be implemented as DCOM (Distributed Component Object Model) objects making possible to share their definitions over a network, thus enabling a distributed computing of the resource requests. The Simulation Agent's task is to wrap around a physical resource in a software entity working in the VL system.

The whole system is separated in three different modules: view, server and simulation module. Every of the modules contains its own set of agents and other objects and communicates with the objects in the other modules via defined protocols that give a possibility to exchange several common messages.

Figure 2 represents the collaboration diagram of the scenario that goes when the system is servicing a resource request. User initiates the scenario by taking an action in the web interface. The Container Agent communicates the action to the View Agent, which tries to connect to the Server Agent in order to send the resource request. The Server Agent is a connections coordinator and it doesn't process the requests itself, instead it runs a Resource Selector Agent that takes its role in the communication between view and simulation modules. After the acceptance of the connection request by the Resource Selector Agent, all of the requests from the View Agent are passed through the Communication Agent and Simulation Agent to the (real) resource. After the resource processes the request and acts accordingly, the Simulation Agent gets the result back and passes up through the Communication Agent and Resource Selector Agent to the View Agent that shows the result to the user, by redrawing it self for example.

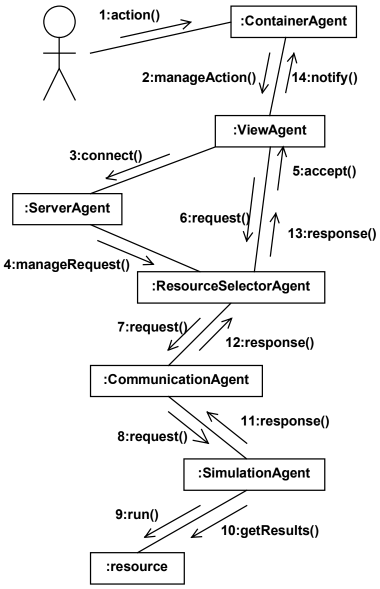


Figure 2 Collaboration diagram

3. Implementation

Our system can run on three hosts. The first host is the client machine of the user browsing the VL via Browser. The second host is the VL server, running web server and Server and Resource Selector Agents. This host is connected to the log and resource definitions databases as well. The third host is connected to the (real) resource and runs the Communication and Simulation Agents.

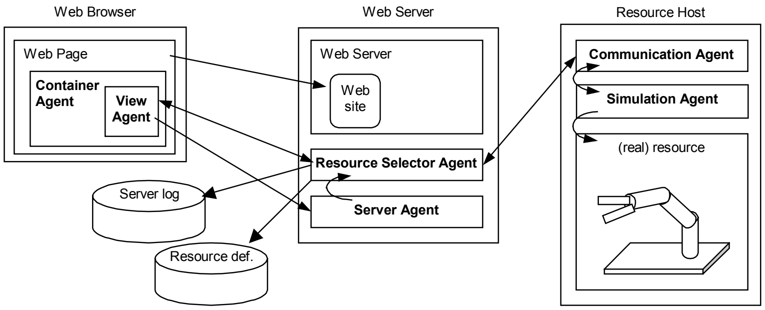


Figure 3 The main components of the VL system

The main components of the VL system are shown on Figure 3.

References

[1] V. Trajkovic, D. Davcev, G. Kimovski, Z. Petanceska, "Web – Based Virtual Classroom", Accepted for presentation in IEEE proc. Of TOOLS USA 2000 Conf., Santa Barbara, California, USA, July 31 – August 3, 2000.

[2] Smith H., Poulter K., "Share the Ontology in XML-based Trading Architectures", Communications of the ACM, Vol. 42, No. 3, (1999).

Application Centric Mobile Agent Systems: Bringing the Focus Back to the Applications

Paulo Jorge Marques, Luís Moura Silva, João Gabriel Silva
CISUC, University of Coimbra, Portugal

{pmarques, luis, jgabriel}@dei.uc.pt

1. Introduction and Motivation

The Mobile Agent (MA) paradigm has now been around for some years. Although there are many advantages associated to the use of mobile agents and many agent platforms are currently available from both the research community and industry, they have not reached a wide acceptance. In fact, mobile agents have gained the reputation of being "a solution looking for a problem" [1], and finding a "killer application" is still on the mind of many practitioners of the technology.

Many reasons are typically pointed out on why the technology is not being so successful as it could. Examples include security problems, lack of proper support for fault tolerance and lack of standards. Although these are important problems, we believe that mobile agent systems are not widely deployed fundamentally due to different reasons. The mobile agent community has been mainly focusing on the agent technology and on the mobility issue, rather than on the support needed for real-world application development. The problem can be viewed in two dimensions: the programmer and the user.

The Programmer

From the point of view of the programmer, constructing an application that uses mobile agents is a difficult process. Current mobile agents systems force the development to be centered on the agents, many times requiring the applications themselves to be coded as a special type of agents – *stationary agents*. When this does not happen, special interface agents (*service agents*) have to be setup between the application and the incoming agents. These agents must know how to speak with the mobile agents and with the application. Although the mobile agent concept – *a thread that can move to another node*, is a very useful structuring primitive, all the currently required setup is an overkill that prevents acceptance by the developers.

Since basically anything that can be done with mobile agents can be done using simple client/server remote method evocations, the reasoning goes: "Mobile agents do not give me any fundamentally different (and needed) mechanism, and at the same time force me to develop systems in a completely different way. Why should I bother to use them?"

The problems include: the mobile agent concept is not readily available at the language level; the applications have to be centered on the mobile agents; and a complicated interface between the agents and the applications must be written. The programmers want to develop their applications as they currently do. Agents will typically play only a small role on the application (90-10 rule: 90% traditional development, 10% mobile agents). Current systems force exactly the opposite.

The User

From the viewpoint of the user, if an application will make use of mobile agents then it is necessary to first install an agent platform. The security permissions given to the incoming agents must also be configured and the necessary hooks to allow the communication between the agents and the application must be setup. While some of these tasks can be automated using installation scripts, this entire setup package is too much of a burden for the average user. Usually, the user is not concerned with mobile agents nor wants to configure and manage mobile agent platforms. The user is much more concerned with the applications than with the middleware they are using in the background. In the currently available mobile agent systems, the agents are central and widely visible. They are not the background middleware but the foreground applications.

Another important issue is that the term "*mobile agents*" has very strong negative connotations that make the dissemination of the technology difficult. The user is afraid of installing a platform capable of receiving and executing code without his permission. This happens even though the existence of mobile code is present in technologies like Java, in particular in RMI and JINI. The fundamental difference is that in those cases, the user is shielded from the middleware being used. In many cases, using mobile agents does not pose an increased security threat, especially if proper authentication and authorization mechanisms are in place. However, because the current agent platforms do not shield the user from the middleware, the risk associated with the technology is perceived as being higher, which causes users to back away from applications that make use of mobile agents.

2. The M&M Approach

In the M&M project at the University of Coimbra, we are working on an approach to overcome some of the problems previously identified. We are developing flexible lightweight JavaBeans software components that when incorporated into an application gives it the capability of sending and receiving agents. The application is developed using industry OO development best-practices and can additionally become agent-enabled by the simple drag-and-drop of mobility components. In our approach, the emphasis is put on the development of applications, not on the agents. Each agent arrives and departs from the application that it is specific. The application knows the interface of the agents and the agents know how to interact with the applications (Figure 1).

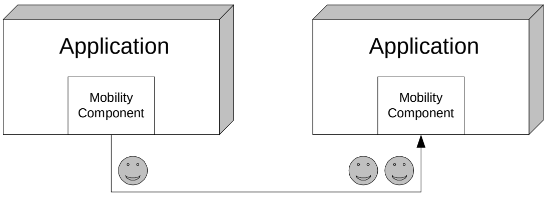


Figure 1 – Applications become agent-enabled by using mobility components.

From the programmer's perspective, all he has to do is to use the mobility components and write the agents. The agents arrive and departure directly from the application without the needing a fully blown agent platform. From the user point of view, he just sees an ordinary application. The usage of mobile agents is completely shielded from him. This is a step forward over the traditional development approaches used in the platform-based MA systems. Because in this approach there is no mobile-agent platform and because the emphasis is put on the application and not on the agents, we call this approach *ACMAS – Application-Centric Mobile Agent System*.

M&M is being implemented using the JavaBeans component model. Nevertheless, we have also decided to support the ActiveX component model, that is based on Microsoft's COM [2]. We have taken this decision because of the wide adoption of this component model in the software industry and in research. We believe that by supporting ActiveX, a much wider usage base and a much richer set of environments where to use our framework will be available. Another important point is that any language that has the necessary mechanisms to make use of COM and ActiveX is able to use the mobility components. At the present, most of the languages that available for the Windows [3] operating system have that capability. By supporting ActiveX, we are no longer limited to develop the applications in Java. We can program the applications in languages like Visual C++, Visual Basic and Delphi. This happens without having to implement any special layers between our components and the target languages. During our experiments, we developed a simple instant messaging application in Visual C++, Visual Basic and Java. The agents migrated and executed in all the client applications independently of the language in which they were written.

At this stage of the project, we already have a rich component palette that in a modular way supports many of the features found in traditionally MA platforms. We have components for mobility support, inter-agent communication, security, agent tracking and infrastructure management. We are currently expanding the component palette into two different application domains: accessing information systems in disconnected computing environments [4] and network management. Our vision is to build an extensive framework where the programmer combines components available off-the-shelf from third-party software producers, components for supporting mobile agents, and components designed specifically for the application domains being considered.

3. Conclusion

We believe that the factors that are preventing a wide adoption of the mobile agent paradigm are not only related to the technological problems that still exist, but to the great overhead that its usage imposes on the programmers and users. When weighting the benefits obtained from using mobile agents against the implications of having them in terms of software development and utilization, the programmers and users choose not to use them.

In the M&M project, we are trying to overcome the limitations identified on the traditional platform-based model by using binary software components. The idea is not to use traditional mobile agent platforms but to embed sufficient support for mobility inside of the applications.

References

[1] *The Future of Software Agents*, Volume 1, Number 4, IEEE Internet Computing, available at http://computer.org/internet/v1n4/round.htm, 1997.

[2] D. Rogerson, *Inside COM*, Microsoft Press, 1996.

[3] "Microsoft Windows Products Homepage," http://www.microsoft.com/windows/default.asp.

[4] P. Marques, L. Silva, J. Silva, *A Flexible Mobile-Agent Framework for Accessing Information Systems in Disconnected Computing Environments*, to be presented at the Third International Workshop on Mobility in Databases and Distributed Systems (MDDS'2000), Greenwich, London, September 2000.

XMILE: An Incremental Code Mobility System based on XML Technologies

Cecilia Mascolo, Wolfgang Emmerich, and Anthony Finkelstein
Dept. of Computer Science University College London
Gower Street, London WC1E 6BT, UK
{C.Mascolo|W.Emmerich|A.Finkelstein}@cs.ucl.ac.uk

Logical mobility ranges from simple data mobility, where information is transferred, through code mobility allows the migration of executable code, to mobile agents, in which code and data move together. Several application domains need a more flexible approach to code mobility than can be achieved with Java and with mobile agents in general. This flexibility can either be required as a result of low network bandwidth, scarce resources, and slow or expensive connectivity, like in mobile computing settings, or scalability requirements like in applications on several thousand clients that have to be kept in sync and be updated with new code fragments.

We show how to achieve more fine-grained mobility than in the approaches using mobile agents and Java class loading. We demonstrate that the unit of mobility can be decomposed from an agent or class level, if necessary, down to the level of individual statements. We can then support incremental insertion or substitution of, possibly small, code fragments and open new application areas for code mobility such as management of applications on mobile thin clients, for example wireless connected PDAs or mobile phones, or more in general distributed code update and management.

This work builds on the formal foundation for fine-grained code mobility that was established in [3]. That paper develops a theoretical model for fine-grained mobility at the level of single statements or variables and argues that the potential of code mobility is submerged by the capability of the most commonly used language for code mobility, i.e., Java. We focus on an implementation of fine-grained mobility using standardized and widely available technology.

It has been identified that mobile code is a design concept, independent of technology and can be embodied in various ways in different technologies. The eXtensible Markup Language (XML) [1] can be exploited to achieve code mobility at a very fine-grained level. XML has not been designed for code mobility, however it happens to have some interesting characteristics, mainly related to flexibility, that allow its use for code migration. In particular, we will exploit the tree structure of XML documents and then use XML related technologies, such as XPath and the Document Object Model (DOM) to modify programs dynamically. The availability of this technology considerably simplifies the construction of application-specific languages and their interpreters.

XML provides a flexible approach to describe data structures. We now show that XML can also be used to describe code. XML DTDs (i.e., Data Type Definition) are, in fact, very similar to attribute grammars. Each element of an XML DTD corresponds to a production of a grammar. The contents of the element define the right-hand side of the production. Contents can be declared as enumerations of further elements, element sequences or element alternatives. These give the same expressive power to DTDs as BNFs have for context free grammars. The markup tags, as well as the PCDATA that is included in unrefined DTD elements, define terminal symbols. Elements of XML DTDs can be attributed. These attributes can be used to store the value of identifiers, constants or static semantic information, such as symbol tables and static types. Thus, XML DTDs can be used to define the abstract syntax of programming languages. We refer to documents that are instances of such DTDs as XML programs. XML programs can be interpreted and in interpreters can be constructed using XML technologies. When XML programs are sent from one host to another we effectively achieve code mobility.

Unlike Java programs, which are sent in a compiled form, XML programs are transferred as source code and then interpreted on a remote host. Unlike Java, XML does not confine us to sending coarse-grained units of code; XML documents do not need to begin with the root of the DTD, they can also start with other symbols of the grammar. This enables us to specify sub-programs and even individual statements. We refer to such code fragments as XML program increments. Hence, we can specify complete programs as well as arbitrarily fine-grained increments in XML. Figure 1 shows how we ship XML programs (every transport protocol can be used). It is possible to specify where to

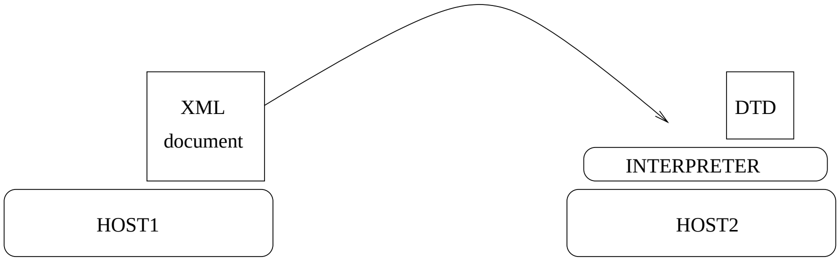


Figure 1: XML Program Migration to Remote Interpreter.

insert/substitute the code increment into the program through the use of XPath for addressing nodes of the tree model defined by the DOM on the XML program. Figure 2 shows the migration of the code increment and the addressing of the insertion point with XPath.

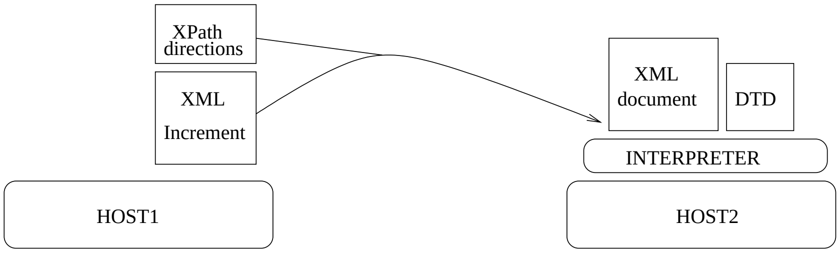


Figure 2: Increment Migration to Robot Site.

The model on which XMILE is based is presented in [2]. We are currently enriching the model and implementing a prototype. We introduced the ability to dynamically bind data and code; single variables as well as single lines of XML code can be moved and patched dynamically into a program, while the XMILE system provides dynamic binding. XML tags for pro-active mobility have been added to the prototype to allow the mobility of lines from the XML programs themselves.

The XMILE system also allows the update of the Java classes used by the XML interpreter (see Figure 1), in order to modify the interpretation of the XML code.

We are now working on mobile computing applications, in particular we used the Symbian EPOC RE5 emulator to run the prototype and we are currently evaluating the performance of XMILE in this setting. Java cards and PDAs are also in the list of devices that we consider for possible application of the approach.

Furthermore work on no-stop (no-reboot) applications can take advantage of XMILE as it is possible to exploit the tree structure to constrain the program execution temporarily to branches of the program while updating other parts (with no need of shutting down the application).

We are now investigating the use of XML schemas in order to allow the modification of the language that for the time being is constrained by the definition of the DTDs. Schemas allow much more flexible treatment of the domain specific languages defined and introduce yet another interesting potential evolution of the XMILE approach.

References

[1] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language. Recommendation http://www.w3.org/TR/1998/REC-xml-19980210, World Wide Web Consortium, March 1998.

[2] W. Emmerich, C. Mascolo, and A. Finkelstein. Implementing Incremental Code Migration with XML. In M. Jazayeri and A. Wolf, editors, *Proc. 22nd Int. Conf. on Software Engineering (ICSE2000)*, pages 397–406, Limerick, Ireland, June 2000. ACM Press.

[3] C. Mascolo, G.P. Picco, and G.-C. Roman. A Fine-Grained Model for Code Mobility . In O. Nierstrasz and M. Lemoine, editors, *Proc. 7th European Software Eng. Conf. (ESEC/FSE 99), Toulouse, France*, volume 1687 of *LNCS*, pages 39–56. Springer, 1999.

Mobile Agent Platform for Mobile Devices

Patrik Mihailescu, Elizabeth A. Kendall and Yuliang Zheng

Peninsula School of Network Computing
Monash University, McMahons Road, Frankston, VIC 3199, Australia
patrik77@bigpond.com{kendall,yuliang.zheng}@infotech.monash.edu.au

1 Extended Abstract

Up until now, mobile agent platforms (e.g. Aglets, Mole, Concordia, etc) have been confined to operate within high-end desktop environments such as Windows, Unix and Solaris. This poster paper presents work in progress in the development of a mobile agent platform for small mobile devices such as cell phones, pagers, home appliances and personal digital assistants (PDA). These devices are becoming more and more popular with users due to their small size and their ability to be used while the user is on go. We believe that these devices will benefit greatly from the use of mobile agent technology. Typically these devices are connected via a wireless network. Wireless networks, compared to wireline networks suffer from lower bandwidth, have a greater tendency for network errors and have a higher cost of network connectivity. Mobile agents will not only help to overcome these network limitations, but they will also help to enhance the level of functionality offered by applications on these devices. For instance, applications will become more independent and able to operate without constant user input. Applications may evolve during their lifetime, i.e. agent locates additional application component and installs them.

We are in the process of developing a mobile agent platform for mobile devices using the Connected Limited Device Configuration (CLDC) specifications, which is part of the Java 2 Micro Edition (J2ME). The CLDC specification is aimed for devices, which contain around 160KB to 512KB of memory and operate with either a 16 or 32 bit RISC/CISC microprocessor. Devices such as cell phones, pagers, personal organizers and point of sales devices fall into the CLDC specification. There are numerous differences between the J2ME and the Java 2 Standard Edition (J2SE)[1]. One of the strengths of the J2ME is its flexible architecture, which allows it to operate on a large number of technically diverse devices. This is achieved through the use of profiles, which allows a particular family of devices (cell phones) to define common functionality required by the underlying Virtual Machine (VM). Therefore different groups of devices (even toasters) may define different profiles containing functionality that is only appropriate for a particular group of devices.

The CLDC specification also utilizes a new VM called the KVM (Kilo Virtual Machine). This has been designed to operate with as little as 128k memory, which includes the Virtual Machine (VM), the class libraries and heap memory for executing

[1] The whitepaper located at http://java.sun.com/products/cldc/wp/KVMwp.pdf covers these differences

applications. We have decided to embed our agent platform directly into the KVM, as opposed to defining additional Java classes (on top of the exiting KVM classes), which will need to be resolved by the KVM when called. Our approach is in contrast to the approach taken by existing mobile agent platforms. The advantages for us are significant, as our classes will be loaded directly into memory when the KVM loads for the first time. Therefore any access to our classes will be substantially quicker as they will no longer need to be interpreted; they are already in native C code. This saving is crucial as the majority of our intended devices operate with relatively low memory/processing/power and any additional translation of Java classes to native methods will cause notable performance problems. To embed our classes within the KVM, we had to rebuild the KVM. We used the romizing approach, by performing all the linking and resolving of our classes using a program known as the JavaCodeCompact. This program will accept as input all our Java classes and after its execution will produce a C representation of every single one of our Java classes. This is used in conjunction with the original KVM C source code and compiled/linked to produce a mobile agent enabled KVM.

The makeup of our agent platform contains several common features located in existing mobile agent platforms, e.g. mobile agents, messaging, events, etc. However there are also significant differences, as we have aimed to provide specific functionality for the intended devices that use our KVM. This functionality falls into four main areas: XML, security, networking and agent tracking. The majority of our classes are written in Java and then translated to C using the JavaCodeCompact program. However we have been forced to write several native methods in particular for our security routines, as the KVM does not provide any Java security classes or any Java math classes. The XML support we are providing is a SAX based non-validating parser, which can be used to interpret XML based messages. As not all devices will have a TCP/IP stack installed, we are providing support for a variety of protocols for all network functions (messages, sending/receiving of agents) such as the IrDA protocol, Bluetooth, Serial and TCP/IP. As the KVM does not provide any security, we are implementing a security scheme based on public key techniques called Signcryption. Finally we are providing support for tracking of agents, with particular emphasis on making sure agents can return safely back to their original location.

At the moment, we are finalizing our Java classes, making sure they match our functional specifications. The next stage of our work will be the development of several agent-based applications, which will help to evaluate and identify both strengths and weaknesses in our architecture. Our agent platform is being tested on a variety of Palm device (Palm Vx, Palm IIIx, Palm IIIc)[2], but once Motorola release their KVM for cell phones we will also be including these devices in both our testing and development. Further information regarding our development work can be located at the following web site `http://www.pscit.monash.edu.au/~patrikm/`.

[2] Currently the KVM only runs on Palm devices

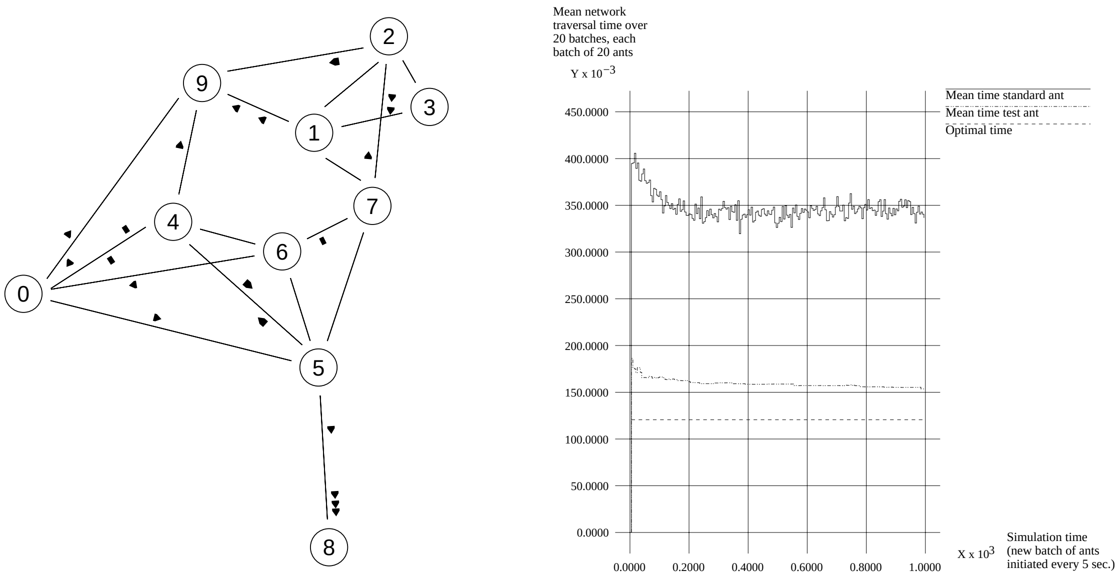
Simulating Mobile Agent Based Network Management using *Network Simulator*

Otto Wittner, Bjarne E. Helvik
{ottow, bjarne}@item.ntnu.no

Department of telematics
Norwegian University of Science and Technology
7491 Trondheim
Norway
http://www.item.ntnu.no/˜ottow

Abstract

Large, heterogeneous computer and telecommunication networks with ever changing topology and size are typical environments todays network management (NM) systems struggle to control. Unexpected events occur frequently in these complex networks, and many of the events result in failure situations where the NM system is required to intervene and restabilize the network. By distributing the NM system throughout the network efficiency and dependability can be improved. This is indeed what todays NM system providers and operators are focusing on. Efforts are currently being made to increase the level of distribution of the most popular management architectures in use today (SNMP, OSI, TMN).



(a) (b)

Figure 1: (a) Surveillance ants traversing a network in active network packets. The illustration is a screen dump from *Network Animator*, an animation tool included in the *Network Simulator* suit. (b) Preliminary results showing how the ants learn to traverse the network by “smelling” each others pheromone tracks. Standard ants move stochastically while test ants always choose the most desirable next hop. The lower line is the optimal traversal time for visiting all nodes and returning home.

Swarm intelligence (i.e. simple mobile software agents with collective behavior) is an alternative concept for implementing distributed applications. Several promising examples of NM applications based on swarm intelligence are under development [1, 5, 4]. For most of these examples a simulator has been used as the major tool to configure system parameters and verify system performance.

This paper describes how *Network Simulator (NS)* has been extended to support simulation of mobile code, and how a network node surveillance system based on swarm intelligence is being developed using *NS*.

NS is an open source simulator package. Development efforts are currently funded by DARPA through the VINT project [2]. *NS* is capable of running realistic simulation scenarios of traffic patterns in IP-based networks. The package is implemented as a neat mix of OTcl and C++ classes.

Simulations involving mobile code can not easily be run using the standard *NS* package. An extension is required to include the necessary features.

DARPAs Active Network (AN) architecture enables mobile code in network environments. AN packets and AN enabled network nodes are very similar to mobile agents and mobile agent systems respectively. AN packets can only contains small units of software. This fits nicely with the concept of swarm intelligence (a large number of small and simple agents).

The PANAMA project (TASC and the University of Massachusetts) has developed an AN extension to *NS*. In the original extension AN packets do not contain any executable code. A reduction in packet size is used to simulate execution of a packet. To enable simulation of swarm based applications we have enhanced the AN extension to handle mobile code by allowing a reference to an OTcl object be inserted as packet data in an AN packet. AN enabled nodes has been extended to initiate a “bring back to life” method (called *run()*) in such referenced objects.

Using the AN enabled *NS* we have started developing and testing a network surveillance system . The system is inspired by how ants performed foraging and is based on work done by Dorigo et al on ant colony optimization (ACO) [3]. The goal of an ant in our system is to visit all nodes in the network and return to its home node, the quicker the better, i.e. the traveling salesman problem (TSP). Dorigo has developed an efficient ACO-algorithm, Ant Colony System, for solving TSPs but the algorithm requires global access to all nodes which is impossible in a real network. Our system relies only on local information. Figure 1 shows some preliminary results. Our ants do indeed learn more efficient traversal itineraries over time but they struggle to find the optimal itinerary.

More work is required to tune the surveillance system. A genetic algorithm is currently being applied for parameter tuning.

Work is also in progress on testing the performance of the surveillance system in a more realistic environment where several other traffic sources generate traffic in the network . *NS* provides a collection of traffic sources which can easily be added to a simulation setup.

Work has just been initiate on a backup-path reservation system based on swarm intelligence. Ants allocate resources to form backup paths for end-to-end connections. Multiprotocol Labeled Switching (MPLS) is chosen as the underlying routing mechanism. *NS* provides support for MPLS simulation.

Our first experiences with *Networks Simulator* as a simulation tool for swarm intelligence based application are promising. Speed of simulation is reasonable if care is taken when deciding which objects to implement in OTcl and C++. *NS* provides functionality for convenient creation of realistic simulation environments.

References

[1] Gianni Di Caro and Marco Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9:317–365, Dec 1998.

[2] DARPA: VINT project. UCB/LBNL/VINT Network Simulator - ns (version 2). http://www.isi.edu/nsnam/ns/.

[3] Marco Dorigo and Gianni Di Caro. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(3):137–172, 1999.

[4] Navarro Varela G. and M.C. Sinclair. Ant Colony Optimisation for Virtual-Wavelength-Path Routing and Wavelength Allocation. In *Proceedings of the Congress on Evolutionary Computation (CEC’99)*, Washington DC, USA, July 1999.

[5] T. White A. Bieszczad B. Pagurek. Distributed Fault Location in Networks Using Mobile Agents. In *Proceedings of the 3rd International Workshop on Agents in Telecommunication Applications IATA’98*, Paris, France, July 1998.