Designing Fault-Tolerant Mobile Systems

Giovanna Di Marzo Serugendo[1] and Alexander Romanovsky[2]

[1] Centre Universitatire d'Informatique, University of Geneva,
CH-1211 Geneva 4, Switzerland
Giovanna.Dimarzo@cui.unige.ch

[2] School of Computing Science, University of Newcastle,
NE1 7RU Newcastle upon Tyne, UK
Alexander.Romanovsky@newcastle.ac.uk

Abstract. The purpose of this paper is to investigate how several innovative techniques, not all initially intended for fault-tolerance, can be applied in providing fault tolerance of complex mobile agent systems. Due to their roaming nature, mobile agents usually run on Java-based platforms, which ensures full portability of mobile code. The first part of the paper discusses specific characteristics of mobile systems, outlines the application areas benefiting from code mobility, and shows why the existing error recovery techniques are not suitable for mobile systems. In the next part of the paper we present evaluation criteria for fault tolerance techniques, and propose several possible solutions for error recovery at the application level: meta-agent, Coordinated Atomic actions, asynchronous resolution, self-repair, and proof carrying code. The intention is to allow system developers to choose the approach which is suited best to the characteristics of the mobile agent application to be designed. To this end we discuss the advantages and disadvantages of each technique, as well as situations in which it provides the most benefit. A simple example, based on Internet shopping, is used throughout the paper to demonstrate the techniques.
Keywords: Mobile agents, system structuring, fault tolerance, exception handling, software engineering.

1 Introduction

From the early 90s the community working on mobile agent systems has been looking for a killer application to prove the usefulness of the concepts. Today, due to the fast development of the Internet, Grid computing, e-commerce, and due to the tremendous growth of the sizes of distributed systems and their proliferation to many new application areas, mobility is becoming a practical issue. Many companies are now building complex applications with some elements of mobility.

In this paper we will focus on code mobility [7], which is mainly used for providing a wide range of non-functional benefits, including on-line system customisation and upgrading, improvement of system performance and dynamic extension of applications. The term mobile agent refers to a software component (including its code and state) that can be moved from one location (node

or host) of a distributed system to another where it will resume its execution. Mobile agent systems are often Internet scale systems. They presume application programmer's awareness of mobility, and cover a large variety of application domains, such as database access, network management or e-commerce [17][1].

Mobile agents are frequently mentioned in literature as being advantageous for helping to cope with faults in distributed systems, since they are able to perform local monitoring tasks, and to deal with network disconnection naturally. The use of mobile agents eases fault-management, especially in large-scale decentralised systems. However, there are still many errors that mobile agent-based applications have to face. The purpose of this paper is not to see how fault-tolerance can be achieved with mobile agents, but how fault-tolerance for mobile agents can be realised.

Mobile agent systems encounter traditional types of error that distributed software does, but also specific errors such as failures of migration requests, violations of security restrictions, specific communication exceptions [24]. The combination of the specific characteristics of mobile agents and the specific errors that they can encounter prevents traditional error recovery techniques from being fully efficient or directly applicable for building mobile applications. Indeed, traditional techniques for fault tolerance in distributed systems:

- are too heavy to be used in mobile systems;
- mainly focus on software tolerance of hardware faults;
- are not applicable for building mobile agents intended for execution on a number of different hosts during their life cycle;
- rely on hard-wired or permanent connections between system components;
- are not applicable for building applications consisting of moving entities scattered around several locations;
- are not suitable for systems whose structure changes dynamically, e.g., changes in system participants, their links and their locations;
- are oriented towards very different execution environments and computation paradigms.

There is a need for much more flexible and dynamic fault-tolerant techniques that are light in both code and communication exchanges. One of the important issues here is to involve application programmers that develop such mobile applications, in providing fault tolerance. We believe that it is inefficient to focus only on fault tolerance that the underlying middleware can provide transparently for the application. Application-specific fault tolerance incorporated at the application level is clearly a much more powerful approach to dealing with faults and abnormal situations of all possible types. In this paper we propose different techniques, not all necessarily meant for fault-tolerance, and show how they could be applied at this level to improve the overall dependability of mobile applications.

[1] Section 2.2 outlines major application domains in which mobile code is used.

2 Mobile Agent Systems

2.1 Characteristics of Mobile Systems

In this paper we consider mobile applications that can be built either as single mobile agents or as a number of co-operating mobile agents. We assume that each mobile agent can decide to move or can be forced to move depending on the application state, and that the agents move by moving both code and current state. Each host consists of a machine and an executing environment (a platform) which provides basic operations for mobility as well as access to local resources. The platform is usually based on a Java virtual machine, which provides full portability of the mobile agent bytecode.

The platform includes typical functions such as move an agent, resolve the name, dispatch, retract, and clone. In case of co-operative agents we assume that the platforms provide means for inter-agent information exchange and/or synchronisation. In this paper we are not focusing on any particular mobile environment as we would like to address general issues common for a number of them.

The most typical characteristics of mobile systems are as follows. These systems are very *dynamic* and *flexible* by nature, so agents join and leave platforms or groups of agents continuously. Usually they are *open systems* whose participants (i.e. the agents) can establish and change connections dynamically. The agents are *autonomous* and may decide to move from one host to another when necessary. This causes the application to be of a *world-wide scale*. Mobile agents usually *communicate* through blackboards *asynchronously*, and *do not know the location* of other agents. Usually a mobile agent application is *decentralised*, i.e., there is no central entity controlling the agents.

2.2 Application Domains

Code mobility is a means for developing flexible, customisable, adaptable, extendable and dynamic services in a number of application domains. In this section we give a brief overview of major domains that (can) benefit from employing mobile agents [11].

Network Management. The use of customised mobile agents for network management increases decentralisation and flexibility. Indeed, agents pro-actively carry out administration tasks, and can be dynamically replaced allowing, for instance, dynamic update of network policies. In addition, since mobile agents are located in network devices, they help reduce traffic around the management station, and make it possible to distribute processing load [1].

Remote Device Control and Configuration. Mobile agents are useful for performing monitoring tasks, such as information filtering, control functions and intrusion detection. The advantage of using mobile agents in this domain is that policies and itineraries of mobile agents can be modified dynamically [23] and can respond to an intrusion in real-time [6].

Active Networks. In an active network, nodes of the network - routers and switches - can perform computations [20]. On the one hand, it allows customised programs - mobile code - to be injected into the nodes of the network, making active network service management possible. On the other hand, packets, passing through routing elements, are no longer passive packets, because they contain small programs that are executed at each node. This approach enables dynamic optimisations, extensions and replacement of protocols [21].

Wireless Applications. Wireless networks suffer from low bandwidth and disconnection errors. Mobile agents help to overcome these limitations, since a mobile agent, roaming the connected network, can still work on behalf of a mobile user, even if the mobile user is disconnected. Dispatching a mobile agent close to a server reduces data transfer among information servers and the wireless device [14].

E-commerce, M-commerce. Electronic commerce (E-commerce) consists in buying and selling goods using electronic transaction processing technologies. Whereas traditional electronic commerce involves two stationary computers, mobile electronic commerce (M-commerce) involves at least one mobile computer. In both E-commerce and M-commerce, mobile agents are particularly useful in roaming electronic market places for discovering products, engaging transactions and negotiations on the user's behalf [5].

Distributed Information Retrieval. Agents, dispatched at remote information sources, can perform local searches, accesses to data and filtering them, eliminating network transfer of intermediate data [3].

Active Documents. Moving data with mobile code leads to the notion of active documents. Early forms consisted in interactive and animated web pages provided by applets running at the client side. Recent works enable complex documents, such as meeting schedules containing mobile code, to mediate communication among the participants, including notification of participants, and observation of workflow rules [8].

Workflow Management. Workflow, or business process, management systems consist in a set of tasks performed in a controlled order to realise a business purpose. Mobile agents embody a workflow by encapsulating the business process logic. Mobile agents permit the introduction or extension of workflows at run-time, since agents can dynamically uptake or replace components. Mobile agent itineraries (tasks and locations) can be altered at run-time, allowing just-in-time workflows to be realised [13].

Grid Computing and Global Computing. Mobile agents are particularly useful for efficiently monitoring distributed computational resources in Grid systems, since they help in coping with Grid large-scale size and discovering data [22]. Mobile code proved to be very useful for exploiting idle computing resources

in a Grid system, for distributing data and computational tasks, for collecting computed results, as well as for charging and billing participants [2].

Emerging application domains. In addition to the domains listed above, experts foresee that application domains such as: IP Telephony, middleware service coding and service composition, are very likely to benefit from mobile code technology [10].

2.3 Abnormal Situations

There is a wide range of abnormal situations which mobile applications can face, which include: *migration requests failures*, *unavailability of resources* at the new location, *security restrictions*, *communication delays*, *failures of components* in the local (host) environment, *users' mistakes*, *agent programmers' mistakes*, *failures of the subsystems* of a complex mobile application to provide the required services, *node crashes*, *network partitioning*, and differences in the host environments on which mobile code is executed. These abnormal situations are mainly caused by:

1. failures of the underlying components it is using (i.e. their inability to deliver the required service);
2. failures of the agents it is co-operating with;
3. the agent own faults.

Type (1) includes a number of situations such as hardware faults of different nature (node crashes, lost or corrupted messages, disconnections), absence or malfunctioning of the required resources (main memory, disk storage, computation time, databases, information the agent is looking for, etc.), all types of errors reported by the underlying platforms (heap overflow, null pointer). Clearly some of these situations can be better tolerated by the underlying support (the platform) itself because this can be done transparently for the agent (e.g. using ordered delivery, group communication, replication) but in the situations when the support is not designed to do this all responsibility for recovery is left with the application. Very often local (i.e. involving only one agent located in this host) recovery can deal with such situations. Errors of type (2) include partner's failures and partner's inability to correctly communicate with the agent. They result in the agent inability to continue its computation and require co-operative handling involving several agents [9]. The abnormal situations of type (3) usually require local recovery at the agent level.

All situations that we have described above require application-specific handling. It is important here that depending on their type we can discuss the structuring issues of which part of the mobile application should be involved in the handling. Each agent first tries to handle all the abnormal situations of types 1 and 3 locally (generally speaking, the application-specific handlers will be developed for handling particular errors). When this recovery is not possible, a higher level recovery is applied, which involves a number of cooperating agents.

In a single-agent application, we should report a failure exception to a specialised component (a master waiting for the results, a guardian, a monitoring system, a client, an operator). When one of the co-operating agents is faulty a co-operative handling should be initiated in all of them. Proper structuring of multi-agent systems will allow us to find a subset of agents that have been recently involved in cooperation to avoid involving all of them in such cooperative recovery each time any of them signals an exception.

2.4 Challenges in Fault Tolerance

Mobile agents requirements with respect to fault tolerance include:

- the need to recover from errors encountered at remote locations;
- the need for light code, which can be easily moved among hosts. This imposes a light recovery scheme, if it is included into the mobile agent;
- the highly (world-wide) distributed nature of mobile systems make it impossible or difficult to maintain checkpoints enabling a group of agents to perform a backward error recovery;
- the highly dynamic nature of mobile systems is hardly compatible with the notion of transactions that forces several agents to stay in the same transaction as long as it does not commit or abort;
- abnormal situations are not known in advance, very often agents, cooperating within an application, are neither designed together nor developed by the same programmers. However, an agent should be robust enough to overcome them and continue its execution;
- very often mobile systems are executed in heterogeneous environments without agreed standards or communication interfaces;
- the location of each agent in a group of agents is unknown at run-time. Raising exceptions, having an immediate effect to stop the execution of agents, is impossible. It necessarily implies some delays;
- mobile agent systems are particularly sensitive to security concerns.

These points show that traditional techniques for fault tolerance in distributed systems are not directly applicable for handling abnormal situations of different types at the agent level. First of all, traditional techniques are mainly oriented on tolerating hardware faults (e.g. by using replication, group communication, transactions); but typically mobile systems have to face faults of a much wider range. Secondly, recovery based on rollback or abort cannot be always applied as the main recovery technique, because there are many application-specific modifications that cannot be simply 'erased'; this is why many abnormal situations have to be dealt with at the application level, using forward error recovery [12]. One more crucial factor for choosing forward error recovery is its comparatively low cost. Thirdly, mobile systems need flexible recovery (exception handling) solutions; in particular, it should be possible to change dynamically system configuration, structure, fault tolerance means to be applied and agent connections. One more reason is that, as we have explained above, there is a

huge variety of abnormal situations of different types that can happen. The last reason is that mobile environments have computational models and communication techniques that are very different from the conventional ones (for example, they use asynchronous communication and dynamic binding).

3 Fault Tolerance Techniques for Mobile Agent Systems

The ultimate aim of this section is to propose a number of innovative ways of providing fault tolerance in mobile systems. All of the techniques discussed below rely on application-level forward error recovery as the most general way of dealing with faults of a widest possible range. The section starts with a brief outline of the criteria used in discussing and comparing these techniques. It then presents them using a working example, and discusses pros and cons of each technique with respect to the identified criteria.

3.1 Evaluation Criteria

The choice of the criteria is driven by our intention: to show the strength of each technique in providing typical fault tolerance activities; to analyse the ease with which it can be applied by the application programmers; and to understand how it fits the specific requirements of the mobile applications.

Structuring Fault Tolerance. As stated before, we focus on fault tolerance at the level of the application, and assume that low-level errors either are tolerated by the middleware components or, when this is not possible, are reported to the application at the appropriate abstraction level. Two structuring approaches can be distinguished here: either the fault tolerance code is separated from the functionality of the mobile entities or it is part of them.

- *Built-in Fault Tolerance.* In the case of a built-in fault-tolerant mechanism each agent is equipped with fault tolerance. Such a strategy enables the agent to be autonomous and independent of any other (e.g. a centralised entity) that is responsible for recovering from errors. The main disadvantage is that the agent is not flexible in providing fault tolerance, so, for example, it cannot tolerate unanticipated errors.
- *Separating Fault Tolerance from Functionality.* The alternative to built-in fault tolerance is provided by decoupling functionality code from error recovery code. The advantages of such a scheme are multiple: the error handler can be downloaded when necessary, it can be adapted to the particular error, several versions may be available, the agent can rely on a dedicated agent that provides recovery. The disadvantages are as follows: there may be some delay to move the error handling code; delegating the responsibility for error detection and correction to other agents may be not applicable for mobile agents that cannot always contact error handling agents; the decentralised nature of the application is compromised if the recovery scheme is in some sense centralised.

The fault tolerance techniques described below either use built-in fault tolerance (Coordinated Atomic actions actions, proof carrying code) or separate it from the functionality (meta-agent), or they allow programmers to flexibly choose these two (asynchronous resolution, self-repair).

Error Processing and Error Confinement. The essence of any fault tolerance technique is its error processing (error detection and error recovery) and error confinement capabilities. This includes the *way the errors are detected* when a particular technique is used, and the *information* that can be used for this detection (for example, internal agent states, input/output parameters of the calls, histories of event).

The area of error confinement defines the *damage domain* to be recovered after an error has been detected - this can be an agent, a part of an agent (e.g. its method) or a group of agents. This criterion is closely related to the structuring view discussed before.

Recursive Structuring and Scalability. Another vital issue is *recursiveness* of the technique: generally speaking, these techniques should allow developers to build recursive systems in which failure to deal with any error at a system level is transformed into an error that has to be handled at a higher system level [19]. This can be provided in different ways: action nesting, sub-agents, slave agents, client/server, action groups/subgroups, etc.

Scalability of a technique is related to the ease in which fault tolerance can be provided in systems consisting of multiple agents. We believe that from the fault tolerance point of view this issue is closely related to recursive system structuring.

Overheads. This criterion is used to compare the application-neutral (hidden) overheads caused by the techniques, which may include: a number of *additional messages* implying additional traffic; the *amount of code* to be moved to provide fault tolerance; etc.

Flexibility, Run-Time Reconfigurability. The flexibility of a particular technique is vital for developing fault tolerance means suitable for mobile applications. Code mobility itself provides a powerful support for implementing fault tolerance.

3.2 Working Example

In this subsection we introduce a small banking system that operates with mobile agents and requires fault tolerance. This example is used to illustrate the discussion of possible fault-tolerant techniques that could be used for mobile systems.

A user, wishing to acquire some product, launches a buyer agent that will roam the Internet searching for a seller agent offering the requested object. The buyer and seller agents will meet in an electronic market place, where they will exchange information regarding products that their respective users desire to buy or sell. Each agent is equipped with an e-purse holding some amount of electronic money. If the two agents have matching requests, they reach an agreement (the product is booked, and a contract is concluded), and the payment is then realised by transferring some money from the e-purse of the buyer agent directly to the e-purse of the seller agent. In this case the e-purse acts as cash money, there is no need to ask for a bank certification. If the payment fails, then the agreement has to be cancelled, and the seller agent releases the product.

In this scenario, the buyer agent can be either alone or composed of several agents roaming the Internet simultaneously. It may be difficult or even impossible for the user to contact the buyer agent or for the buyer agent to contact the agents distributed world-wide. Indeed, a mobile IP scheme enabling to contact mobile agents irrespectively of their position is difficult to consider when agents neither know each other nor participate in the same application.

We will consider the following errors in this scenario:

- there is no offer matching the request;
- there is a bug in the seller agent code: it is impossible to reach an agreement or to pay money from its e-purse;
- the buyer agent e-purse does not contain sufficient money;
- the buyer agent e-purse does not present sufficient privileges, e.g., a configuration error does not authorise money to be withdrawn from the e-purse.

3.3 Meta-Agent

In the meta-agent scheme, the normal functionality is contained in the agent, while the fault-tolerant or resource control aspect is left to the meta-agent [25].

In our electronic market place, each agent has an additional meta-agent, as shown by figure 1. We will now consider the errors listed above. If the buyer agent does not find any corresponding seller, it informs its meta-agent that will take the appropriate decision: stop the buyer agent, let it move to another place, inform the user for changing the request, etc.

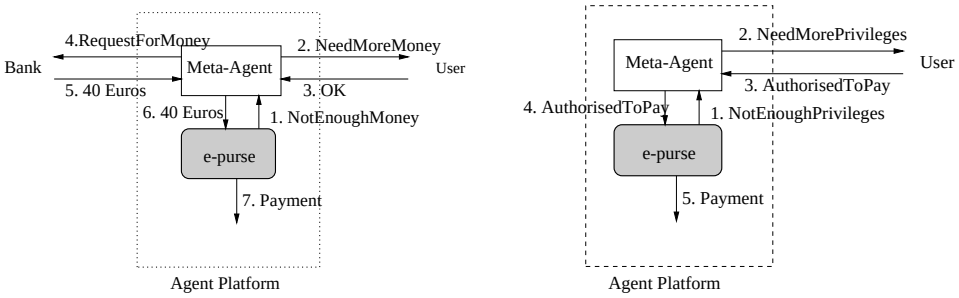
The bug in the seller code results in an internal error for the buyer agent. It will abort its current transaction, and try to find another partner. Alternatively, it may inform the meta-agent about the problem.

In the case (a) when the buyer has not enough money, the buyer agent raises an exception to its meta-agent (1), which is responsible for downloading money to the buyer agent e-purse. The meta-agent first contacts the user (2) for his agreement (3), and then the bank (4) for actually loading money (5), and finally uploads the electronic money to the e-purse (6). The payment can then occur (7).

For insufficient privileges (b), the meta-agent acts in a similar way, it simply asks the user for more privileges, and changes the code of the agent accordingly (4).

Discussion. Meta-agents offer the advantage that the error handling code (actually contained in the meta-agent) can be downloaded or changed at run-time, thus favoring flexibility and adaptability with respect to errors. More than one meta-agent can be related to an agent, thus enabling several exception handling. Meta-agents may either come with the agent or be only requested in case of problems or simply be stationary at some well-known place. There is no overhead at the level of the agent itself, even though at the level of the application, there is a certain amount of code dedicated to the meta-agent, and some message transfer is necessary between the agent and its meta-agent.

The meta-agents are useful for an asynchronous recovery from local errors (i.e., the meta-agent can deal with the error and then inform the agent), when there is no need for a cooperative resolution scheme at the level of the agents and when exceptions are raised in isolated agents. This may be the case for wireless applications and active networks.



(a) Insufficient Money (b) Insufficient Privileges

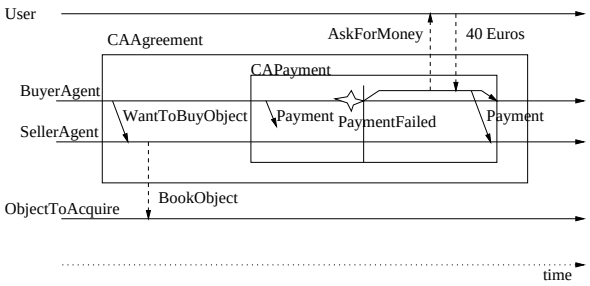
Fig. 1. Meta-Agent

3.4 Coordinated Atomic Actions

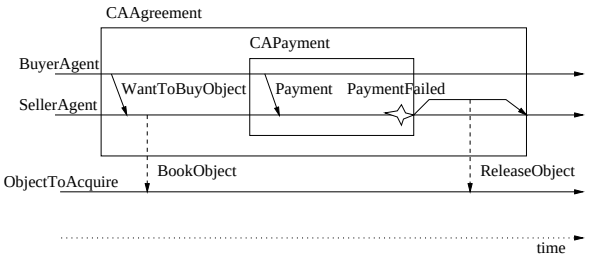
Coordinated Atomic actions (CA actions) [26] are a structuring mechanism for developing dependable concurrent systems. CA actions generalise the concepts of atomic actions and transactions. Atomic actions control cooperative concurrency among a set of participating entities (e.g., processes, objects) and provide forward error recovery using cooperative exception handling. Transactions maintain the coherency of external resource competitively accessed by concurrent actions. CA actions can be nested and when an action is not able to provide the required result an exception is propagated to the containing action.

Regarding the market place example, figure 2 shows a buyer and a seller agents entering a CA action, called CAAgreement, for realising the agreement. This CA action contains a nested CA action for the payment, CAPayment.

Whenever the payment fails, either (a) the buyer agent enters a recovery phase, where it asks for more money, and the payment finally succeeds; or (b) the buyer cannot recover, leading to the failure of CAPayment, which in turns causes CAAgreement to abort. We notice that before aborting CAAgreement releases the booking of the object that had been previously realised.



(a) Recovery Succeeded



(b) Abort

Fig. 2. Coordinated Atomic Actions

Discussion. CA actions offer the most general way of providing fault tolerance in concurrent systems: they clearly define the damage area (i.e., the CA action scope) to be recovered and are intended for recursive system structuring. The hidden overheads are caused by additional synchronisation of the agents participating in an action on the action entry and exit.

CA actions are especially well suited to complex applications involving cooperating agents; for large-scale applications in which agents are aware of other agents, i.e., active documents; for systems in which we need cooperative handling of exceptions, i.e., workflow management applications or when we need recursive structuring of complex mobile applications for fault tolerance. However, additional work has to be carried out to make this concept more applicable for mobile applications. This in particular includes better understanding of the underlying

principles of agent participation in a cooperative activity, requires introducing location-independent and location-aware actions, as well as open actions, and developing a specialised support based on the existing Java RMI support [27].

3.5 Asynchronous Resolution

In some cases, several roaming agents act collaboratively for the resolution of a given task. If the task has to be aborted, all the agents have to be informed in order to stop their work. However, those agents are unattainable by asynchronous exceptions, since we do not know where they are.

Consider, for instance, the scenario in which we want to cancel purchasing a series of stamps, where each stamp in the series is bought by an individual agent, possibly located in different market places. This problem can be solved using an asynchronous resolution scheme, depicted by figure 3: individual agents, from time to time, may consult some agreed place (AgentPlatform_3), where messages for the agents are stored. Either this place contains no more money that the agent can use, and thus they will no longer be able to buy stamps; or it contains directives to stop buying stamps, and if possible to re-sell those that have already been purchased.

This collaborative asynchronous resolution scheme may also help the buyer agents to recover from money/privileges problems, since one of them may ask the others for more money or for additional privileges.

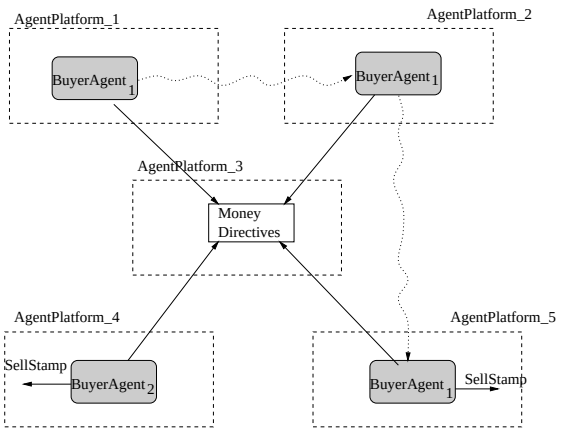


Fig. 3. Asynchronous Resolution

Discussion. This approach naturally scales to large-size systems because the fault detection and the recovery solutions employ the mobile agent mechanisms (blackboard communication, and mobility). The flexibility of the recovery mechanism and the overhead depend on the way the agent code related to the

recovery mechanism is managed. If additional meta-agents are used, then flexibility increases and the agent can cope with new directives, in this case the additional code is stored in meta-agents. Otherwise, flexibility is limited to the original code present in the agent.

Asynchronous resolution is well suited to asynchronous recovery (i.e., when there is no need for all the agents to recover at the same time) and for collaborative agents in the situations when they can be individually recovered from the same exception in their particular locations. Grid computing and e-commerce are good candidates for this kind of fault tolerance technique.

3.6 Self-Repair

In this case, the agent does not consult its user (neither through a meta-agent, nor directly). It tries instead to solve the problem autonomously by asking other agents, not necessarily involved in the same task, for help.

For instance, in the case of the electronic market place, if the buyer does not find any matching offer, it may ask other agents if they know about possible sellers.

As shown by figure 4, if the buyer agent has no sufficient money (1), and provided it is authorised to spend more money than the e-purse holds, it can ask all other agents present in the market place to borrow it some money (2). If an agent agrees to lend money to the buyer agent (3), they conclude a contract (specifying interests and delays). Then, the buyer agent receives some money, that it will use for buying the initial object (4). The agent that lent the money will be refunded afterwards, once the buyer agent will have downloaded some money directly from its bank. This scheme can be very convenient in case of urgent decisions. Borrowing money from another agent present in the market place may be realised more quickly than entering a remote communication with the bank.

Discussion. It is worth noting that this kind of resolution can be undertaken with meta-agents as well as with CA actions. However, it suits better for the situations: when the agent is given full autonomy or crucially needs to recover even partially from abnormal situations, e.g. wireless applications; when control of the application is decentralised both for the functionality and the fault tolerance requirements; and for situations requiring a quick resolution, i.e., remote device control or active networks.

This technique scales to large-size systems since the error is confined to the agent itself, which autonomously tries to solve the problem. Obviously there is an overhead in the size of code necessary to carry resolution schemes and in execution time needed to overcome problems.

3.7 Proof Carrying Code

Before trying to reach an agreement and then realising that the buyer has not enough money, the proof carrying code alternative enables both parties to expose

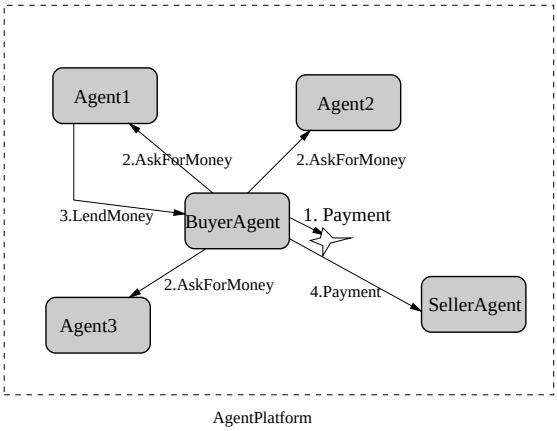


Fig. 4. Self-Repair

some of their internal information, that will prove or not that they are able to enter the transaction. In the example of figure 5, the buyer and seller agent exchange information regarding the amount of money contained in the e-purse, the privileges they have (buy or sell), as well as the minimum and maximum price requested or allowed for the object. Given the values, we see that the buyer has not sufficient money (only 4 Euros), even if an agreement is reached for a price between 5 and 7 Euros.

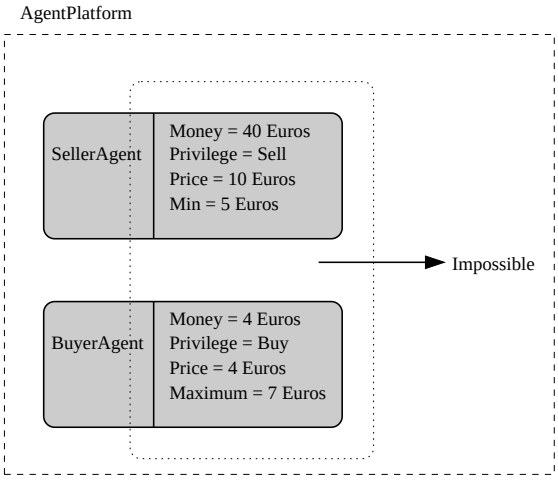


Fig. 5. Proof Carrying Code

In the case of bugs in the seller agents, an additional proof carried by the seller agent code may allow for them to be detected.

This is a simplified version of the original proof carrying code [16]. We can easily imagine how to actually replace specification values with code.

Discussion. Flexibility of this approach is limited since the proof, used for error detection, is part of the agent and not customisable at run-time. There is a clear overhead in code size due to the proof that has to be carried by the agent and in execution time required to execute it.

This scheme is particularly well adapted for mobile agents that enter interactions with unknown agents, i.e., agents that have been designed independently and that may use different standards, and for discovering and composing unknown services proposed by potential partners.

4 Related Works

The main body of research in fault tolerance of mobile systems focuses on software techniques oriented towards tolerating hardware faults. For example, an approach to developing consensus protocols to be used for achieving an agreement when agents crash is presented in [18]. A novel algorithm for ensuring that a message is always delivered to a mobile agent is proposed in [15]. The particularity of this algorithm is that it does not enforce continuous connectivity with the message source. Our intention is to build on such techniques when possible to allow tolerance of faults of a wider range. The techniques we are considering usually use such services as the underlying middleware supports (which promotes separation of concerns). Moreover, when some of them are either not capable of delivering the required service or not used or very expensive to use, the agents can deal with such problems in an application-specific fashion using the techniques discussed in Section 3.

There are two techniques which are related to this work. Paper [9] introduces an agent service (called Electronic exception handling institution) dedicated to detecting and solving exceptional conditions in software agent marketplaces. This service collects information about typical abnormal situations and uses AI techniques to find out the best way of handling them. It is interesting that the authors recognise the needs for involving several agents in cooperative handling of complex exceptions (in a way similar to CA actions). In the scheme proposed in [24] each mobile agent has a guard agent - comparable to meta-agents - that handles all exceptions propagated from it. This scheme is very important, as it is the first scheme dealing with the specific characteristics of the mobile environment.

5 Conclusion

Mobile agent systems have specific characteristics (they are highly decentralised, dynamic and made of roaming entities) and encounter specific abnormal situations (migration errors, resource and security problems, partner's failures). These

particularities impose requirements for fault tolerance that traditional techniques hardly succeed in meeting. This paper discusses several approaches to providing fault tolerance, at the application level, in the framework of mobile agent systems. It is worth noting that the same ideas could be well applied to other decentralised systems, such as biologically inspired ones, and to applications running on portable devices and facing physical mobility. Our future work includes proposing a scheme combining proof carrying code and light specifications for agent programming, developing a flexible support for open mobile CA actions and establishing patterns of exception handling for Lana [4], a mobile agent platform incorporating a support for security, coordination and disconnection of nodes.

6 Acknowledgment

Alexander Romanovsky is partially supported by European IST DSoS (Dependable Systems of Systems) Project (IST-1999-11585). Giovanna Di Marzo Serugendo is supported by Swiss NSF grant 21-68026.02.

References

1. M. Baldi, S. Gai, and G. P. Picco. Exploiting code mobility in decentralized and flexible network management. In K. Rothermel and R. Popescu-Zeletin, editors, *Proceedings of the 1st International Workshop on Mobile Agents 97 (MA'97)*, volume 1219 of *LNCS*, pages 13–26. Springer-Verlag, 1997.
2. W. Binder, G. Di Marzo Serugendo, and J. Hulaas. Towards a Secure and Efficient Model for Grid Computing using Mobile Code. In *8th ECOOP Workshop on Mobile Object Systems: Agent Applications and New Frontiers*, June 2002.
3. B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko, and D. Rus. Mobile Agents for Distributed Information Retrieval. In M. Klusch, editor, *Intelligent Information Agents*, chapter 15, pages 355–395. Springer-Verlag, 1999.
4. C. Bryce, C. Razafimahefa, and M. Pawlak. Lana: An Approach to Programming Autonomous Systems. In *16th European Conference on Object-Oriented Programming, ECOOP'02*, 2002.
5. P. Dasgupta, N. Narasimhan, L. E. Moser, and P. M. Melliar-Smith. MAgNET: Mobile Agents for Networked Electronic Trading. *IEEE Transactions on Knowledge and Data Engineering, Special Issue on Web Applications*, 11(4):509–525, July-August 1999.
6. N. Foukia, S. Hassas, S. Fenet, and J. Hulaas. An Intrusion Response Scheme: Tracking the Source using the Stigmergy Paradigm. In *Proceedings of Security of Mobile Multiagent Systems Workshop (SEMAS-2002)*, 2002.
7. A. Fuggetta, G. P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, 1998.
8. F. Kilander, P. Werle, and K. Hansson. Jima - A Jini-based Infrastructure for Active Documents and Mobile Agents. In *Proceedings of the Personal Computing and Communication (PCC) Workshop*, November 1999.
9. M. Klein and C. Dellarocas. Exception handling in agent systems. In O. Etzioni, J. P. Müller, and J. M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 62–68. ACM Press, 1999.

10. D. Kotz, R. Gray, and D. Rus. Future Directions for Mobile Agent Research. *IEEE Distributed Systems Online*, 3(8), 2002.
11. D. B. Lange and M. Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, 1999.
12. P. A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice*. Dependable computing and fault-tolerant systems. Springer-Verlag, 1990.
13. S. W. Loke and A. B. Zaslavsky. Towards distributed workflow enactment with itineraries and mobile agent management. In J. Liu and Y. Ye, editors, *E-Commerce Agents, Marketplace Solutions, Security Issues, and Supply and Demand*, volume 2033 of *Lecture Notes in Computer Science*, pages 283–294. Springer-Verlag, 2001.
14. Q. H. Mahmoud. MobiAgent: A Mobile Agent-based Approach to Wireless Information Systems. In *Proceedings of the 3rd International Bi-Conference Workshop on Agent-Oriented Information Systems, held with the 5th International Conference on Autonomous Agents 2001*, 2001.
15. A. L. Murphy and G. P. Picco. Reliable communication for highly mobile agents. *Journal of Autonomous Agents and Multi-Agent Systems*, 5(1):81–100, March 2002.
16. G. C. Necula. Proof-carrying code. In *The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'97)*, 1997.
17. G. P. Picco. Mobile agents: An introduction. *Journal of Microprocessors and Microsystems*, 25(2):65–74, April 2001.
18. S. Pleisch and A. Schiper. Modeling fault-tolerant mobile agent execution as a sequence of agreement problems. In *19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, pages 11–20. IEEE Computer Society Press, 2000.
19. B. Randell. Recursively structured distributed computing systems. In *Proceedings of Third Symposium on Reliability in Distributed Software and Database Systems*, pages 3–11. IEEE Computer Society Press, 1983.
20. D. L. Tennenhouse. Active networks. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation (OSDI '96)*, pages 89–90, Berkeley, CA, USA, 1996. USENIX Association.
21. D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 1997.
22. O. Tomarchio, L. Vita, and A. Puliafito. Active monitoring in GRID environments using mobile agent technology. In *2nd Workshop on Active Middleware Services (AMS'00) in HPDC-9*, August 2002.
23. A. Tripathi, T. Ahmed, S. Pathak, A. Pathak, M. Carney, M. Koka, and P. Dokas. Active Monitoring of Network Systems using Mobile Agents. In *Proceedings of Networks 2002, a joint conference of ICWLHN 2002 and ICN 2002*, 2002.
24. A. Tripathi and R. Miller. Exception handling in agent-oriented systems. In A. Romanovsky, C. Dony, J. Lindskov Knudsen, and A. Tripathi, editors, *Advances in Exception Handling Techniques*, volume 2022 of *LNCS*, pages 128–146. Springer-Verlag, 2001.
25. A. Villazon and W. Binder. Portable Resource Reification in Java-based Mobile Agent Systems. In *Proceedings of the Fifth IEEE International Conference on Mobile Agents (MA'01)*, December 2001.
26. J. Xu, B. Randell, A. Romanovsky, C. Rubira, R. Stroud, and Z. Wu. Fault tolerance in concurrent object-oriented software through coordinated error recovery. In *25th International Symposium on Fault-Tolerant Computing Systems (FTCS-25)*, pages 499–509. IEEE Computer Society Press, 1995.

27. A. F. Zorzo and R. J. Stroud. A Distributed Object-Oriented Framework for Dependable Multiparty Interactions. *ACM Sigplan Notices*, 34(10):435–446, October 1999.