10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07), pp. 141-145 (2007)

On Rigorous Design and Implementation of Fault Tolerant Ambient Systems

Alexei Iliasov, Alexander Romanovsky, Budi Arief
School of Computing Science, Newcastle University
Newcastle upon Tyne NE1 7RU, England
{Alexei.Iliasov, Alexander.Romanovsky, L.B.Arief}@newcastle.ac.uk

Linas Laibinis, Elena Troubitsyna
Department of Information Technologies, Aabo Akademi University
20520 Turku, Finland
{llaibini, etroubit}@abo.fi

Abstract

Developing fault tolerant ambient systems requires many challenging factors to be considered due to the nature of such systems, which tend to contain a lot of mobile elements that change their behaviour depending on the surrounding environment, as well as the possibility of their disconnection and re-connection. It is therefore necessary to construct the critical parts of fault tolerant ambient systems in a rigorous manner. This can be achieved by deploying formal approach at the design stage, coupled with sound framework and support at the implementation stage. In this paper, we briefly describe a middleware that we developed to provide system structuring through the concepts of roles, agents, locations and scopes, making it easier for the developers to achieve fault tolerance. We then outline our experience in developing an ambient lecture system using the combination of formal approach and our middleware.

1 Introduction

Ambient systems and applications are now used in various critical domains, such as health, transport, emergency, and production systems. Many of these systems will rely on the *mobile agent paradigm*, which supports structuring systems using decentralised and distributed entities cooperating to achieve their individual aims. These systems have a number of characteristics complicating their development and making it difficult for the developers to meet stringent requirements. Firstly, a vast majority of emerging ambient systems and applications have mobile elements, such as code, devices, data, services and users. Secondly, such systems need to be context-aware, so that the system activities can be directly influenced by the information representing their changing environment (due to the component mobility or to changing characteristics of the physical world in which the systems are executed). Thirdly, these systems are open, in a sense that components can appear and disappear (e.g., become disconnected). Therefore, developers of such systems need certain abstractions and middleware for supporting component mobility, context-awareness, and system openness. In addition to these, due to the large number of components and the decentralised nature of these systems, the developers need to ensure system flexibility and scalability.

Fault tolerance is becoming a very important aspect of these systems due to (i) their complexity and openness, which make it impossible to avoid or remove faults altogether; (ii) a high heterogeneity of components, modes of operation and requirements, all of which can cause architectural mismatches; (iii) non-professional users; (iv) a high level of reliance society puts on such systems, trusting them to perform a wide range of everyday critical functions: these systems are rapidly becoming business- and safety-critical; and (v) the expected wide proliferation of these systems and the impact this will have on our society.

Unfortunately, the existing frameworks for developing mobile, context-aware and open ambient systems do not provide adequate support for achieving fault tolerance. The main difficulties here are due to component (agent) mobility and their autonomy. Mobility creates new challenges for ensuring system fault tolerance [11] as – in addition to all kinds of faults found in sequential, concurrent and distributed systems – mobile agents are susceptible to a number of unique faults and potentially-harmful situations due to recurrent disconnections and re-connections. Agent autonomy and anonymity, asynchronous communication, and system openness also call for the development of novel structuring mechanisms as the existing agent systems do not support recursive system structuring using units of agent cooperation, which can serve as units of error confinement and system recovery.

Fault tolerance mechanisms for ambient systems can be created at different levels: hardware, operating system, middleware or application. In our work, we are focusing on both the middleware and the application levels fault tolerance, supporting mainly forward error recovery through the systematic use of exception handling [6]. Although there have been several mechanisms proposed for general message-passing systems (such as [12] and [13]), this area has not received enough attention in the context of coordination-based systems even though it can provide the most efficient and effective recovery from a wide variety of errors and faults such as environmental faults/deficiencies, component, architectural and organisational mismatches and system-level inconsistencies, potentially damaging changes in components and environments, and degradation of component services.

In our previous work, we introduced a *Context-Aware Mobile Agents* (CAMA) system [2], which provides fault tolerance in mobile agent applications through agent structuring. This is achieved by using the concepts of roles and scopes, explicit and consistent exception handling at scopes level, and a specialised distributed middleware for detecting disconnections and raising exceptions at the scope level.

The overall aim of our research is to propose and evaluate a rigorous method for developing fault tolerant ambient applications [15]. Introducing the CAMA abstractions and the accompanying middleware is part of this work. In addition to this, we are working on applying formal methods and tools in order to ensure a clean system design and an early verification of the system models [8, 10]. The main reason for this is that it is becoming clear for us that applying advanced fault tolerance mechanisms is itself a complex task that often results in misuse and mistakes (see some supporting evidence in [3] and [14]). To help in avoiding them – where possible – we investigate the use of formal fault tolerance methods focusing specifically on employing adequate tool support and modelling fault tolerance patterns.

The rest of the paper is organised as follows. Section 2 outlines the CAMA system – including its main abstractions and support for fault tolerance, and how agents can be constructed using the CAMA system. Section 3 briefly shows how an ambient application can be built through the combination of formal modelling and the CAMA system. We are using an ambient campus lecture scenario [16], the development of which is part of the IST RODIN project [15]. Section 4 concludes the paper and outlines our future work.

2 Context-Aware Mobile Agents (CAMA)

CAMA is both a framework and a middleware supporting the development and deployment of agent-based applications. As a framework, CAMA encourages disciplined development of fault-tolerant mobile agent applications by supporting a set of abstractions ensuring system structuring, exception handling and openness. We have implemented this framework as a middleware that can be used for supporting effective and highly scalable mobile applications, while guaranteeing agent compatibility and dependability. This section provides a brief introduction to CAMA – a more detailed description can be found in [2]. The full implementation of the middleware and its adaptation layer (called `jcama`) are available at SourceForge [7].

2.1 CAMA Abstractions

The three basic concepts which CAMA offers for system structuring are *agent*, *platform* and *location*. Agents represent the basic structuring unit in CAMA applications and they are the active entities of a CAMA system. Each agent is executed on a platform; several agents may reside on a single platform. A platform provides an execution environment for agents, as well as an interface to the middleware. A platform is typically run on a PDA, a smartphone, or a laptop. A location is the core part of any CAMA system as it acts as the middleware that provides a means for communication and coordination among agents which are situated within the range of the location middleware (connections are typically conducted over wireless networks, with wireless hotspots providing access to the location middleware).

A location is also a container for *scopes*. A scope structures the activity of agents in a specific location and provides an isolation of several communicating agents, thus structuring the communication space. Scopes are dynamically created when the entry conditions defined in the scope specification are met. Agents can cooperate only when they are participating in the same scope. Nested scopes are used to structure large multi-agent applications into smaller parts which do not require the participation of all agents. Such structuring has a number of benefits. It isolates agents into groups, thus enhancing security. Scope structuring is also crucial for developing fault tolerant applications as it links the coordination-space structuring with activity structuring, which supports error confinement, localised error recovery and scalability.

To deal with various functionalities that any individual agent provides, CAMA introduces agent *role* as a finer unit of code structuring. Each agent has one or more roles associated to it. A role is a specification of one particular functionality of an agent. A composition of all agent roles forms its specification. An agent participates in a scope by assuming one of the roles available for that scope. The scope definitions include specifications of the roles from which the scope is composed. The role specifications determine the roles available in the scope, and the number of agents allowed to take part under any given role in that scope. In other words, a role is a structuring unit of an agent, and it is an important part of the scoping mechanism. It allows a dynamic composition of multi-agent applications and ensures agent interoperability by enforcing the developers of roles and agents to conform to the role specifications.

2.2 Design for Fault Tolerance

The CAMA framework supports application-level fault tolerance by providing a set of abstractions and a supporting middleware that allow developers to design effective error detection and recovery mechanisms. The main means for implementing fault tolerance in CAMA is a novel exception handling mechanism which associates scopes with the exception contexts. Scope nesting provides recursive system structuring and error confinement, as information cannot be passed outside such scopes. In effect, the execution of a scope is atomic from an outside observer's point of view.

Error recovery in CAMA systems is application-specific by nature and is to be implemented by the role developers. Error recovery is typically conducted at the level of individual scopes with an aim to recover the activity of this scope, although it is possible to attach handlers to individual roles (we usually do not use this feature as it breaks the abstraction levels). CAMA allows the developers to define cooperative recovery involving some or all roles of a scope when an error is detected in this scope. After detecting an error, any role can initiate application-specific recovery at the scope level.

A rich set of predefined exceptions provided by CAMA is useful for writing applications which react to abnormal situations detected by the CAMA middleware (Fig. 1). There are two types of abnormal situations: the ones which are propagated to all scope roles which are subscribed to them (including connection-disconnection exceptions, such as `CamaExceptionDisconnection`) and the local exceptions propagated to an individual role when it tries to execute an illegal action (e.g., violation of the scope constraints exceptions, such as `CamaExceptionClosed`).

Events	*Exceptions*
NewScope	CamaExceptionClosed
Destroy	CamaExceptionInvalidReqs
Join	CamaExceptionNoRights
Leave	CamaExceptionInvalidRole
Disconnection	CamaExceptionDisconnection

Figure 1. Some system events and predefined exceptions

A number of predefined middleware events allow an agent to track contextual changes, most importantly, changes in the set of visible agents and scopes. This is essential for initiating both cooperative and localised recovery. For example, after discovering a disconnection of another agent, an agent may initiate local recovery actions that put it into a state from which it can continue without the disconnected agent.

It is our ongoing work to make the approach initially presented in [8] more suitable to developing fault tolerant open multi-agent applications. Compared with the work outlined in [8], our current framework imposes less restrictions on the agents during exception handling, in particular, an agent does not have to be involved in exception handling at all, if this does not suit its aims. This makes exception handling not only anonymous and asynchronous, but also voluntary, making it very different from the classical atomic action schemes (such as that proposed in [4]).

Many researchers realise now that fault tolerance is becoming a software engineering concern which needs to be addressed at various development steps. Finding the right balance between using early and late development step techniques is a difficult issue. This paper shows how formal models can be used in conjunction with implementation level techniques. Formal modelling and verification typically help in eliminating a number of errors that otherwise would have to be addressed at the implementation stage. As part of our work, we investigate (i) how error detection and recovery can be integrated into formal development, (ii) how formal models can be used by extracting from them information about undesirable behaviour to incorporate error detection and recovery actions in the implementation, and (iii) how recovery can be introduced at the level of agent construction. When fault tolerance is integrated into formal models, it becomes an integral part of the system, so that fault tolerance properties are verified and satisfied during system development. In order to use formal models for incorporating fault tolerance into system implementation, we need to define the undesirable behaviour as an action or a set of actions which break the model invariant or one of the post-conditions. This helps a developer to include, at the implementation step, an additional code for recovering from the undesirable behaviour. The formal approach we are using defines a set of roles which are interoperable by construction. During system implementation, agents are constructed as configurations of several roles. This approach clearly requires agent-level error detection and recovery to be introduced during system implementation.

2.3 Agent Construction

A typical CAMA agent is composed of a number of simple building blocks. The overall structure of a CAMA agent is shown in Fig. 2. The discovery part is responsible for finding a location and connecting to it. Once an agent is connected to a location, it decides which application scopes to join or to create. An agent can have *physical mobility* (due to the physical movement of the hosting device) and *logical mobility* (when it changes its hosting platform). Any non-trivial agent has a monitor which oversees its context, which changes during both physical and logical mobility. The agent actions responsible for migration are put into a separate part. There are also the implementations of agent roles and the units for coordinating the roles.

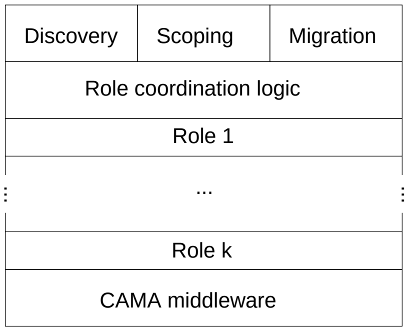


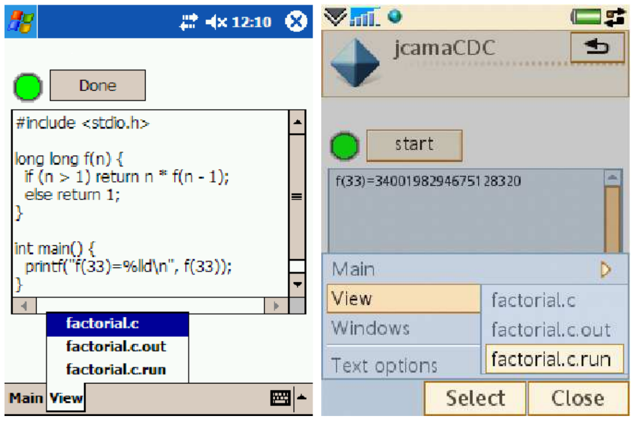
Figure 2. Agent subcomponents

The bulk of the services provided by the middleware is implemented as calls to the location service. Hence, the first action of an agent is to associate itself with one or more locations. There are two mechanisms for this: the automated location discovery based on UDP multicasting and the use of a fixed location address. The middleware also provides a simple connection method which takes a location service address as its argument. This way of connecting is not very useful in dynamic environments, for example when an agent physically moves with a PDA owner. During the lifetime of an agent, a set of available locations changes and there may be no way of determining their addresses. To overcome this problem, the middleware provides a multicast-based discovery mechanism. The mechanism supports local-area networks such as LAN, WLAN and BluetoothPAN.

3 Ambient Campus Lecture Scenario

We picked the ambient campus lecture scenario described in [16] for showing how formal method can be used in combination with the CAMA system for developing robust and dependable ambient applications.

We use B [1] to carry out the formal modelling of the ambient campus lecture application. The B method supports the top-down development paradigm. In the development process, the abstract specification is transformed into a system implementation via a number of correctness-preserving steps called *refinements*. Refinements allow us to gradually incorporate concrete implementation details, while at the same time preserving the previously stated properties of



```
#include <stdio.h>

long long f(n) {
  if (n > 1) return n * f(n - 1);
  else return 1;
}

int main() {
  printf("f(33)=%lld\n", f(33));
}
```

```
f(33)=3400198294675128320
```

Figure 3. Screenshots of the student agent

the system. The correctness of each refinement step is validated by proofs. As a result, we get an executable system that is correct by construction.

The tools support available for B – for example, Atelier B [5] – provides some assistance to the entire development process. Atelier B has facilities for automatic verification and code generation, as well as documentation, project management and prototyping.

We started with an abstract specification of the ambient lecture system, and through the refinement process, we incorporated implementation details concerning concrete functionality, communication and fault tolerance aspects of the involved agents. More detailed discussion on our formal modelling work can be seen in [9].

In this scenario, we assume that each classroom is a location with wireless support, in which a lecture can be given. Our aim is to develop a system supporting a number of functionalities to be conducted by the teacher and the students during a lecture. The teacher software is run on a desktop PC available in the classroom, while the student software is run on a mobile device such as PDA or smartphone. During the lecture, the students can be involved in a group work where the teacher – through the application's graphical user interface – puts the students into several groups. The groups are mutually exclusive, i.e., a student cannot belong to more than one group. Within each group, the students work collaboratively on a given task. As an example shown in this paper, the students are asked to develop C programs. An important part of this application is a subsystem that supports *shared editing*, allowing multiple students to modify the same piece of C code in a consistent manner within a distributed setting.

We implemented the whole case study application in Java using the features provided by `jcama`. The application was deployed on two PDAs, one smartphone, and two desktop PCs. One of the PCs was used for hosting the location middleware whereas the other was used by the teacher agent. Students can move about during the lecture, capitalising on the wireless connectivity of their devices.

Fig. 3 shows the screenshots of the student agent. The left screenshot shows the shared editor, running on a PDA. Only one student is allowed to modify the C code contained in the shared editor at any one time. The right screenshot is taken from a smartphone and it shows the result of executing the C program through the student agent.

The PCs run linux-2.6 and JDK-1.5. The PDAs run Windows Mobile 2003 SE and IBM J9 Java machine, connected to the location middleware using a wireless LAN infrastructure. The smartphone is a SonyEricsson M600i running Symbian 9.1 with ad-hoc Bluetooth networking.

4 Conclusions and Future Work

The main contribution of this paper is in introducing a novel approach for developing fault tolerant ambient applications by using a combination of a formal method augmented with specialised development patterns and a set of design abstractions supported by a dedicated middleware. This approach has been successfully applied in developing the lecture scenario part of a larger ambient campus system.

We found formal methods to be very useful in allowing us to clearly define and rigorously develop in a stepwise fashion the most critical part of the application. Our experience suggests that it is useful to combine formal methods with the more commonly-used ways of building systems. In our work of developing the lecture scenario, we have identified and applied several ways of using them in combination.

Our ongoing and future work focuses on: (i) finalising the set of abstractions and the functionality of the middleware; (ii) building the complete development method supporting – in addition to the B refinement – verification (by model checking) of system properties with a specific focus on the fault tolerance properties; (iii) extending the exception handling mechanism with an ability to involve several scopes, to explicitly state and dynamically modify the exception propagation policies and to use exceptional events (reactions) to further separate normal system behaviour from the abnormal one.

5 Acknowledgements

This work is supported by the IST RODIN Project [15]. A. Iliasov is partially supported by the ORS award (UK).

References

[1] J. R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 2005.

[2] B. Arief, A. Iliasov, and A. Romanovsky. On Using the CAMA Framework for Developing Open Mobile Fault Tolerant Agent Systems. In *Proceedings of Software Engineering for Large-Scale Multi-Agent Systems (SELMAS) Workshop at ICSE 2006*, pages 29–36, 2006.

[3] M. Bruntink, A. van Deursen, and T. Tourwé. Discovering Faults in Idiom-Based Exception Handling. In *Proceedings of ICSE 2006. 20-28 May 2006. Shanghai, China*, pages 242–251. ACM Press, 2006.

[4] R. Campbell and B. Randell. Error Recovery in Asynchronous Systems. *IEEE Transactions on Software Engineering*, 12(8):811–826, 1986.

[5] ClearSy. Atelier B: The industrial tool to efficiently deploy the B method. *http://www.atelierb.societe.com/index_uk.htm*, Last accessed: 24 Nov 2006.

[6] F. Cristian. Exception Handling. In T. Anderson, editor, *Dependability of Resilient Computers*, pages 68–97. Blackwell Scientific Publications, 1989.

[7] A. Iliasov. Implementation of Cama Middleware. *http://sourceforge.net/projects/cama*, Last accessed: 24 Nov 2006.

[8] A. Iliasov and A. Romanovsky. Structured Coordination Spaces for Fault Tolerant Mobile Agents. In C. Dony, J. L. Knudsen, A. Romanovsky, and A. Tripathi, editors, *Advanced Topics in Exception Handling Techniques*, pages 182–201. LNCS-4119, 2006.

[9] A. Iliasov, A. Romanovsky, B. Arief, L. Laibinis, and E. Troubitsyna. On Rigorous Design and Implementation of Fault Tolerant Ambient Systems. Technical report, CS-TR-993, School of Computing Science, Newcastle University, UK. October, 2006.

[10] L. Laibinis, A. Iliasov, E. Troubitsyna, and A. Romanovsky. Formal Approach to Ensuring Interoperability of Mobile Agents. Technical report, CS-TR-989, School of Computing Science, Newcastle University, UK. October, 2006.

[11] G. D. Marzo and A. Romanovsky. Designing Fault-Tolerant Mobile Systems. In N. Guelfi, E. Astesiano, and G. Reggio, editors, *Scientific Engineering for Distributed Java Applications International Workshop, FIDJI 2002, Luxembourg, LNCS 2604*, pages 185–201. Springer, 2003.

[12] R. Miller and A. R. Tripathi. The guardian model and primitives for exception handling in distributed systems. *IEEE Trans. Software Eng.*, 30(12):1008–1022, 2004.

[13] S. Pears, J. Xu, and C. Boldyreff. Mobile agent fault tolerance for information retrieval applications: An exception handling approach. In *ISADS*, pages 115–122. IEEE Computer Society, 2003.

[14] D. Reimer and H. Srinivasan. Analyzing Exception Usage in Large Java Applications. In *Proceedings of ECOOP 2003 Workshop on Exception Handling in Object-Oriented Systems*, 2003.

[15] Rodin. Rigorous Open Development Environment for Complex Systems. *IST FP6 STREP project, http://rodin.cs.ncl.ac.uk/*, Last accessed: 24 Nov 2006.

[16] E. Troubitsyna, editor. *Rodin Deliverable D18: Intermediate Report on Case Study Development*. Project IST-511599, School of Computing Science, Newcastle University, UK, 2006.