Project no.: **IST-FP6-STREP-26979**

Project full title: **Highly dependable ip-based networks and services**

Project Acronym: **HIDENETS**

Deliverable no.: D5.2

Title of the deliverable: Preliminary testing framework and methodology

Contractual Date of Delivery to the CEC:	31st December 2007.
Actual Date of Delivery to the CEC:	21st December 2007.
Organisation name of lead contractor for this deliverable:	LAAS
Author(s):	Hélène Waeselynck[1], Zoltan Micskei[2], Minh Duc N'Guyen[1], Nicolas Rivière[1]
Participants(s):	(1) LAAS, (2) BME
Work package contributing to the deliverable:	WP5
Nature:	R
Version:	3.0
Total number of pages:	49
Start date of project:	1st Jan. 2006 **Duration:** 36 month

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)
Dissemination Level

PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

This Deliverable lays the basis for the testing framework that is being developed within Hidenets. It thus contributes to the following project goal:

"Identify development tools and mechanisms like design patterns and testing methodologies to assist the implementation of said service qualities." (from "Hidenets – Description of Work")

To achieve this goal, a review of relevant literature has been performed together with a testing case study that allowed us to gain concrete insights into validation problems. Work has then been directed toward the definition of a scenario-based testing framework that covers (1) the definition of a language that describes interaction scenarios in mobile settings, and (2) some automated support to analyze and implement scenarios.

Keyword list: Testing, mobile computing systems, scenarios, graph matching.

Table of Contents

Bibliography

[Barton and Vijayaragharan 2003] J.Barton, V. Vijayaragharan. Ubiwise: A Simulator for Ubiquitous Computing Systems Design, Technical report HPL-2003-93, Hewlett-Packard Labs, 2003.

[Baumeister et al. 2003] H. Baumeister et al. “UML for Global Computing”. Global Computing: Programming Environments, Languages, Security, and Analysis of Systems, GC 2003, LNCS 2874, Springer-Verlag Berlin Heidelberg, 2003, pp. 1-24

[Briand et al. 2006] L. Briand , Y. Labiche and J. Leduc, “Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software”, IEEE Trans. on Software Engineering, 32(9), Sept. 2006, pp. 642-663.

[Cardelli and Gordon 2000] L. Cardelli and A. D. Gordon. “Mobile Ambients”, Theoretical Computer Science, Special Issue on Coordination, D. Le Métayer Editor. Vol 240/1, Jun. 2000. pp 177-213.

[Cavalli et al. 1999] A. Cavalli, D. Lee, C. Rinderknecht, and F. Zaidi, “Hit-or-jump: An algorithm for embedded testing with applications to IN services”, in Proc. IFIP Int. Conf. FORTE/PSTV '99, Elevier, 1999.

[Cavalli et al. 2004] A. Cavalli et al., “A validation Model for the DSR protocol”, Proc. of the 24th Int. Conf. on Distributed Computing Systems Workshops (ICDCSW’04), IEEE CS Press, Japan, Mar. 2004, pp.768-773.

[Choffnes and Bustamante 2005] D. R. Choffnes and F. E. Bustamante. “An Integrated Mobility and Traffic Model for Vehicular Wireless Networks”, Proc. of the 2nd ACM International Workshop on Vehicular Ad Hoc Networks (VANET), ACM Press, Germany Sep. 2005, pp 69-78.

[Christian and Schmuck 1995] F. Cristian and F. Schmuck, “Agreeing on Processor Group Membership in Timed Asynchronous Distributed Systems”, UCSD Technical report, CSE95-428, 1995.

[Dai 2005] Zhen Ru Dai, "UML 2.0 Testing Profile", in Broy, M.; Jonsson, B.; Katoen, J.-P.; Leucker, M.; Pretschner, A. (Eds.), Model-Based Testing of Reactive Systems, Springer Verlag, ISBN: 978-3-540-26278-7, 2005.

[Damm and Harel 1999] W. Damm and D. Harel. “LSCs: Breathing life into message sequence charts”, in Proc. of the 3rd Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS’99), Italy, Feb. 1999.

[De Bruin et al. 2004] de Bruin, D.; Kroon, J.; van Klaverem, R.: Nelisse, M.. Design and test of a cooperative adaptive cruise control system, Intelligent Vehicles symposium, pp.392-396, IEEE CS Press, 2004

[Fernandez et al. 1996] J.-C. Fernandez, C. Jard, T. Jéron, G. Viho, “Using on-the-fly Verification Techniques for the Generation of Test Suites”, in Proc. Conference on Computer-Aided Verification (CAV'96), LNCS 1102, Springer Verlag, August 1996, pp. 348-359.

[Guennoun 2006] M. K. Guennoun. “Architectures Dynamiques dans le Contexte des Applications à Base des Composants et Orientés Services”, PhD Thesis. University of Toulouse III, France 2006.

[Grabowski et al. 1994] J. Grabowski, D. Hogrefe and R. Nahm, “Test Case Generation with Test Purpose Specification by MSCs”, in Proc. 6th SDL Forum (SDL'93), North Holland, 1993.

[Grassi et al. 2004] V. Grassi, R. Mirandola and A. Sabetta. “A UML Profile to Model Mobile Sytem”, UML 2004, LNCS 3273, Springer-Verlag Berlin Heidelberg, 2004, pp.128-142.

[Harel and Maoz 2006] David Harel and Shahar Maoz, Assert and negate revisited: modal semantics for UML sequence diagrams, Proc. of SCESM '06, Shanghai, China, 2006.

[Hartman and Nagin 2004] A. Hartman, K. Nagin, “The AGEDIS tools for model based testing”, in Proc. ACM/SIGSOFT Int. Symp. on Software Testing and Analysis (ISSTA 2004), ACM, 2004, pp. 129-132.

[Hidenets D1.1] Hidenets Consortium, D1.1 Use case scenarios and preliminary, public deliverable, URL: http://www.hidenets.aau.dk/Public+Deliverables, 2006.

[Hidenets D4.1.2] Hidenets Consortium, D4.1.2 Evaluation methodologies, techniques and tools (final version), public deliverable, URL: http://www.hidenets.aau.dk/Public+Deliverables, 2007.

[Huang et al. 2004] Q. Huang, C. Julien, and G. Roman. Relying on Safe Distance to Achieve Strong Partitionable Group Membership in Ad Hoc Networks, *IEEE Transactions on Mobile Computing* 3, 2 (Apr. 2004).

[ITU 1999] Z. 120 ITU-TS Recommendation Z. 120: Message Sequence Chart (MSC). ITU-TS, Geneva, Sept. 1999.

[Kerbrat et al. 1999] A. Kerbrat, T. Jéron, R. Groz, “Automated test generation from SDL specifications”, in Proc.of the 9th SDL Forum (SDL’99), Elsevier, June 1999, pp. 135-152.

[Koch et al. 1998] B. Koch, J. Grabowski, D. Hogrefe, and M. Schmitt, ”Autolink- A Tool for Automatic Test Generation from SDL Specifications”, Proc. of the IEEE Int. Workshop on Industrial Strength Formal Specication Techniques (WIFT98), USA ,Oct. 1998, pp. 114-125.

[Kugler et al. 2007] H. Kugler, M.J. Stern and E.J.A. Hubbard, “Testing Scenario-Based Models”, in Proc. Fundamental Approaches to Software Engineering (FASE '07), LNCS 4422, Springer, 2007, pp. 306-320.

[Ladani et al. 2005] B.T. Ladani, B. Alcalde, A. Cavalli, “Passive testing – a constrained invariant checking approach”, in Proc. 17th IFIP Int. Conf. on Testing of Communicating Systems (TestCom 2005), LNCS 3502, Springer, 2005, pp. 9-22.

[Leung et al. 2004] K. R.P.H Leung, Joseph K-Y Ng and W.L. Yeung. “Embedded Program Testing in Untestable Mobile Environment: An Experience of Trustworthiness Approach”, Proc. of the 11th Asia-Pacific Software Engineering Conference (APSEC’04), IEEE CS Press, Korea, Dec. 2004, pp.430-437.

[Lime] Lime, Middleware for mobile applications, URL: http://lime.sourceforge.net/

[Micskei et al. 2006] Z. Micskei, H. Waeselynck, M. D. Nguyen, and N. Riviere. “Analysis of a group membership protocol for Ad-hoc networks,” LAAS Technical Report *no. 06797*, November 2006.

[Morla and Davies 2004] R. Morla and N. Davies, “Evaluating a Location-Based Application: A Hybrid Test and Simulation Environment”, IEEE Pervasive computing, Vol.3, No.2, Jul.-Sep. 2004, pp.48-56.

[Noudem and Viho 2005] F. N. Noudem and C. Viho, "Modeling, Verifying and Testing the Mobility Management in the Mobile Ipv6 Protocol", Proc. 8th Int. Conf. on Telecommunications (ConTEL 2005), Vol.2, IEEE CS Press, Croatia, Jun. 2005, pp. 619-626.

[OMG 2005] Object Management Group (OMG), "UML 2.0 Testing Profile", V1.0, July 2005.

[OMG 2007] Object Management Group, UML 2.1.1 Superstructure Specification, URL: http://www.omg.org/technology/documents/formal/uml.htm, 2007.

[Open Group] The Open Group, TETware, URL: http://tetworks.opengroup.org/

[Pickin and Jézéquel 2004] S. Pickin and J-M. Jézéquel, " Using UML sequence diagrams as the basis for a formal test description language", in Proc. of 4th International Conference on Integrated Formal Methods (IFM2004), LNCS 2999, Springer, 2004, pp. 481-500.

[Sanmiglingam and Coulouris 2002] K. Sanmiglingam and G. Coulouris. "A Generic Location Event Simulator", UbiComp 2002, LNCS 2498, Springer-Verlag Berlin Heidelberg, 2002, pp. 308-315.

[Satoh 2003] I. Satoh. "A Testing Framework for Mobile Computing Software", IEEE Transactions on software Engineering, Vol.29, No.12, IEEE CS Press, 2003, pp. 1112-1121.

[Schroth et al 2005] C. Schroth et al, "Simulating the traffic effects of vehicle-to-vehicle messaging systems", Proc. 5th Int. Conf. on ITS Telecommunications (ITST 2005), France, Jun. 2005.

[Sengupta and Cleaveland 2006] B. Sengupta and R. Cleaveland, "Triggerred Message Sequence Charts", IEEE Trans. on Software Engineering, 32(8), Aug. 2006, pp. 587-607.

[Waeselynck et al. 2007] H. Waeselynck et al. "Mobile Systems from a Validation Perspective: a Case study", Proc. of the 6th International Symposium on Parallel and Distributed Computing (ISPDC'07), IEEE CS Press, Austria, Jul. 2007.

[Willcock et al. 2005] C. Willcock, T. Deib, S. Tobies, S. Keil, F. Engler, S. Schulz, An Introduction to TTCN-3, Wiley, 2005.

Abbreviations

GMP	Group Membership Protocol
LSC	Live Sequence Charts
MANET	Mobile Ad-hoc Network
MSC	Message Sequence Charts
PCO	Points of Control and Observation
SDL	Specification and Design Language
SUT	System Under Test
TMSC	Triggered Message Sequence Charts
TTCN-3	Testing and Test Control Notation Version 3
U2TP	UML 2.0 Testing Profile
UML	Unified Modeling Language
V&V	Verification and Validation

1 Executive summary and Introduction

Task 5.2 of WP5 aims to define a testing framework for mobile-based applications. It thus contributes to the following project goal:

"Identify development tools and mechanisms like design patterns and testing methodologies to assist in the implementation of said service qualities." (from "Hidenets – Description of Work")

Work is focused on the verification of the highest layers in the Hidenets architecture, that is, the application layer and possibly some high-level middleware services. We consider functional (black-box) approaches to test whether applications fulfil their expected requirements. Note that quantitative evaluation, e.g., reliability or availability assessment, is not addressed here (it is studied in WP4). Our interest is on the correctness issue.

As a first step, a review of relevant literature has been performed together with a testing case study that allowed us to gain concrete insights into validation problems. Work has then been directed toward the definition of a scenario-based testing framework that covers (1) the definition of a language that describes interaction scenarios in mobile settings, and (2) some automated support to analyze and implement scenarios.

Accordingly, the structure of the Deliverable is as follows.

Section 2 introduces the specificities of mobile computing systems. These specificities are illustrated by two example applications:

- a group membership protocol (GMP) providing a location-aware membership service in the ad hoc domain.
- A distributed black box application, which is one of the Hidenets use cases.

Section 3 reports on the state-of-the-art on testing. It first addresses testing "traditional" distributed systems, and then considers mobile computing systems which, as was explained in Section 2, are distributed systems with challenging specificities.

Section 4 reports on the case study we performed to gain concrete insights into validation issues. The GMP example was chosen, for which a published specification and an open-source implementation were available. The protocol has been analyzed by reviewing the specification and the code, and then by testing the implementation. The outcomes have been presented in a conference paper [Waeselynck et al. 2007]. They provide us with hints for research direction, and justify orientation of our work toward the scenario-based testing framework that is now developed within Hidenets.

Section 5 presents the testing framework. It uses graphical scenario descriptions to support various test-related activities, such as the representation of requirements, of test purposes, of test cases, or of execution traces. After a general overview, we investigate two issues:

- The language to describe scenarios. Usual scenario languages lack concepts to properly account for mobile settings. We propose extensions to fill this gap. The usefulness of these extensions is exemplified by scenarios coming from both the GMP and distributed black box applications.
- Automated support for the analysis and implementation of scenarios. As the extended scenario language considers the spatial relationships of nodes as first class concepts, we

show that graph algorithms, and more specifically algorithms for graph matching problems, are relevant to support most of the processings defined in the framework. A tool development is on-going, and implementation issues are discussed.

Section 6 concludes with future direction on both the language definition and the tool support. As regards the language, the syntax and the semantics still need to be consolidated, which should be achieved within the next three months. We will then investigate the possibility of defining an UML profile for the language, in close interaction with Task 5.1. As regards automated support for processing scenario descriptions, a key issue is to be able to recognise the occurrences of a target scenario in a (possibly large) execution trace. This comes with computational complexity problems, and work will be devoted to the assessment and improvement of the algorithms being developed.

2 Specificities of mobile computing systems

The term mobility may be related to two different notions. The first one is the mobility of code, yielding *mobile computation*. This kind of logical mobility is not considered here. The second one is the mobility of devices (handset, PDA, laptop, intelligent car…) that move within some physical areas, while being connected to networks by means of wireless links (Blue-tooth, IEEE 802.11, GPRS…). This kind of physical mobility yields *mobile computing*, and is the target of Hidenets. Mobile computing systems, and more specifically systems with applications in the ad hoc domain, can be distinguished from "traditional" distributed systems by the following aspects:

- *Dynamicity of the system structure*. The number of mobile nodes is not fixed. It varies over time, due to the dynamic creation, suspension or shutdown of nodes. For example, users can turn on/off, change operating mode of their devices, to reduce connection costs, save battery or avoid accidental connectivity loss. Besides that, connectivity between nodes is also highly dynamic. As the nodes are free to move arbitrarily, they can join or leave the system in an unpredicted manner, links may be established or destroyed, yielding an unstable connection topology. The topological changes may be constrained by a mobility model (e.g., vehicles move in one-way or two-way direction, speed is bounded…).
- *Communication with unknown partners in local vicinity*. In mobile ad hoc networks, a natural communication is local broadcast. It is used as a basic step for the discovery layer in mobile-based applications (for example, group discovery service for membership protocols, a route discovery in routing protocols…). In this class of communication, a node broadcasts a message to its neighbors. As the topology of the system is unknown, the sending node does not *a priori* know the number and identity of potential receivers. Whoever is at transmission range of the sending node may listen and react to the message.
- *Context awareness*. Each node should have an explicit definition of context, of policies to update the context and to react to contextual changes. The context includes any detectable and relevant attribute of a device, of its interaction with other devices and of its surrounding environment at an instant of time. For example, the context can be information collected by means of physical sensors such as location, time, speed of vehicle, or it can be information about network parameters, such as bandwidth, delay and connection topology. Due to mobility, the context is continuously evolving, so that mobile applications have to be aware of, and adapt to, the induced changes.

To more clearly understand the above challenges, typical applications designed for the mobile ad-hoc domain were analyzed. The main purpose of the analysis was to identify the gaps in the current languages, tools and methodologies and how they could be enhanced to support the mobility aspect. We investigated with the following applications.

GMP: A Group Membership Protocol (GMP) based on safe distance [Huang et al. 2004]. The aim of a traditional GMP is to maintain a consistent view of who is in the group, in spite of the faults that may affect some nodes. This particular GMP was designed for mobile systems where the group membership is constantly changing. With the concept of safe distance, joining a group is only allowed when the nodes are close enough to properly finish the communication before any disconnection, and the group has to be split if nodes move away from each other beyond this distance. Using the concept of safe distance, a protocol was proposed that is intended to (i) provide

a consistent view of group membership, and (ii) prohibit unannounced leaving of the group. In such a service the following non-trivial problems have to be addressed.

- *Dynamicity*: in a fixed distributed systems group changes occur because of node failures, which hopefully are rare events. However in mobile systems hosts move out of each other's range, wireless network communication is unreliable, thus changes are frequent events that have to be managed in a fast and consistent way. The number of participating nodes is unknown at design time, very few constraints can be made when specifying the system's properties.
- *Communication*: broadcasting messages is an essential operation of the protocol, as in other distributed applications. However, broadcast in mobile networks are limited to a certain range, and the current topology has to be consulted to see who will receive a certain message.

Distributed black box application: A "traditional" black box ([Hidenets D1.1], Section 3.2.7), can record relevant data such as: engine / vehicle speed (typically 5 seconds before impact), brake status (again here, 5 seconds before impact), throttle position(s), and even the state of the driver's seat belt switch (on/off). The combination of this information along with other engineering factors is indeed very valuable for motor vehicle accident investigation.

A typical scenario for the distributed black box application is described as follows:

- Car A collects its current contextual data with timestamps (local or global time). It may also collect additional information from the neighboring vehicles (e.g., vehicles at single-hop communication range).
- Car A distributes this information to its neighbors according to some back-up policy. When Car A or its neighbors have access to the infrastructure, the black-box data is also backed up to an Infrastructure Server.
- In case of accident investigation, it is possible to retrieve the data from the infrastructure, or from the other cars.

This example application exhibits some characteristics of mobile computing systems:

- *Broadcast communication in local vicinity*: When a car needs to collect contextual information from its neighbors, it will need to broadcast a request to all other cars in its radio transmission range.
- *Context awareness:* When a car distributes black-box data to its neighbors, its policy may be based on contextual parameters: number of neighbors willing to cooperate, speed, direction, and location of these neighbors. Moreover, the presence of an infrastructure access point should be detected, so that the back-up is preferably made on a fixed server when possible.

The GMP and distributed black-box are used as example applications for our testing framework. A detailed study of the GMP will be presented in Section 4. In Section 5, the testing framework will be exemplified by test scenarios coming from both the GMP and the distributed black-box application.

3 State of the art

Testing is the most widespread verification technique. It consists of executing a system and supplying input values. The outputs provided by the system in response to the inputs are then analyzed to determine a pass/fail *verdict*. The smallest part of testing is called a *test case*, a *test suite* being composed of several test cases. A test case usually addresses some *test purpose*, e.g., it activates a target functionality to be checked. Determining a verdict from the test outputs is known as the *test oracle* problem.

This section first reports on the state-of-the-art of testing "traditional" distributed systems (§3.1). Mobile computing systems are then considered (§3.2), and it is shown that their specificities raise new challenging issues.

3.1 State of the art in testing traditional distributed systems

Distributed systems exhibit specificities that concern their design and validation: concurrency, non-determinism, communication delays, as well as the fact that there is a priori no consistent view of the global state nor a common time reference. As a result, testing a distributed system raises difficult controllability and observability issues.

From a technological viewpoint, this means that complex test architectures may be required. A system is accessed by means of distributed Points of Control and Observation (PCOs) to which test components are attached. Synchronisation procedures are required to coordinate those components. Typically, programming test experiments may involve the development of numerous low-level, platform-dependent scripts. In order to alleviate this task, testing frameworks have been developed. For example, TETware (Test Environment Toolkit) [Open Group] is a multi-platform framework that offers an API for usual test functions. Recently, TTCN-3 (Testing and Test Control Notation Version 3) [Willcock et al. 2005] has emerged as a powerful language dedicated to the specification and implementation of test cases. Compared to its previous versions, TTCN-3 has improved capabilities to accommodate complex test architectures and procedures. It may be expected that the language will receive increasing acceptance in a variety of application domains, well outside its original focus (communication protocols).

At the conceptual level, graphical scenario languages have proven quite useful to support test-related activities. Typical examples of languages are UML sequence diagrams or Message Sequence Charts (MSCs) [ITU 1999] and their derivatives [Damm and Harel 1999, Sengupta and Cleaveland 2006]. They are widely used to represent scenarios of interaction in distributed systems. Indeed, the representation of scenarios may serve different purposes: elicitation of requirements [Kugler et al. 2007], specification of test purposes (that is, of interaction patterns to be covered by testing) [Grabowski et al. 1994], design of test cases [Pickin and Jézéquel 2004], or analysis of execution traces [Briand et al. 2006]. Their popularity is due to their user-friendly syntax, which facilitates communication while opening the door for formal treatments (this however requires that precise semantics are given to the used notations).

Model-based test generation has been extensively investigated in the context of protocol testing. It requires the existence of a complete formal specification. In the protocol community, a widely used specification language is SDL (Specification and Design Language) and there are a number of test generation tools from SDL models, both commercial [Koch et al. 1998, Kerbrat et al. 1999] and academic [Fernandez et al. 1996, Cavalli et al. 1999, Tretmans and Brinksma 2002]. Generally

speaking, the generation proceeds by setting a set of test purposes, and by exploring the specification behavior so as to exhibit test cases that fulfil the purposes. The purposes may be associated with specification coverage criteria (e.g., transition coverage), but most often they are manually given (e.g., using the MSC notation) and represent pieces of behavior that are deemed important to be tested. The generated test cases include verdict assignments and are typically produced in the TTCN language. The test generation technology is now being transferred to UML behavior models (see e.g., [Hartman and Nagin 2004]), and is thus expected to get broader application. However, it is worth noting that such formal approaches are mainly used to tackle software units and protocols, not complex distributed systems.

Passive testing is the most applicable approach in the case of complex systems. In contrast to traditional (active) approaches, it does not introduce any special test inputs. Rather, it consists in monitoring the input and output events of a running system. The recorded trace is then analyzed to check for violation of properties. Passive testing can be seen as complementary to active approaches, since it allows observing a larger sample of behavior patterns than some predefined test cases. The checked properties are typically invariant properties that may, or may not, be derived from a formal model of behavior (see e.g., [Ladani et al. 2005]).

3.2 State of the art in testing mobile systems

As explained in the introduction, mobile computing systems are distributed systems with some specificities: dynamicity of the system structure, communication in a local vicinity, context awareness. Such characteristics have an impact on test-related issues, both from a technological (Section 3.2.1) and from a conceptual (Sections 3.2.2) viewpoint.

3.2.1 Test platforms for mobile computing systems

It is certainly desirable that testing be as realistic as possible with respect to the operational conditions. For example, in [Leung et al. 2004], a telephony application is tested by having human operators carrying handsets in an urban area. In [De Bruin et al. 2004], testing a car-to-car application involves three prototype vehicles driven on a road. However, it is obvious that such kind of testing can only be limited, because of its high costs, of controllability and observability problems (see e.g., [De Bruin et al. 2004]), or even of safety problems (e.g., the experimental car scenarios should not endanger the safety of drivers). In practice, a major part of the testing activities will preferably be performed using emulation/simulation facilities.

For mobile computing systems, the provided facilities should accommodate both the wireless technology and the mobility of nodes.

One approach is to simulate the physical movement by the logical movement of mobile agents. Or in other words, mobile computing is simulated by mobile computation. This approach is adopted in the Flying Emulator [Satoh 2003]. Each mobile device is emulated by a mobile agent that carries the application across the network. This allows the application to be connected to various local servers as if the device had moved across physical areas offering wireless access to these servers. Note that the approach requires that the application be developed in a specific middleware framework. Also, ad-hoc networking is not considered.

A more usual approach is to simulate wireless communication (in both infrastructure and ad hoc modes) by integrating a network simulator as a part of the test platform. Such simulators have been originally developed to support networking research, but they are now also used to experiment with the application level. For example, the ns-2 simulator is used in [Morla and Davies 2004] to

evaluate a health-monitoring application, and in [Schroth et al 2005] to evaluate a car-to-car messaging system. In both cases, the network simulator is only part of the complete platform (see Fig. 1), which also includes a context controller. The context controller is used to simulate contextual information in mobile settings, like location-based data. The application exploits contextual information to take different actions adaptively. Context is also needed by the network simulator to set up parameters for imitating the real network characteristics.

There have been several toolkits for simulating contexts in recent years. Some of them are built on 3D game engines and serve to simulate a first-person view of the physical environment, like Ubiwise [Barton and Vijayaragharan 2003]. Car-to-car applications may use traffic simulators (e.g., STRAW [Choffnes and Bustamante 2005]) that simulate the movement of vehicles along roads. There are also generic location event simulators, like the GLS tool developed at Cambridge [Sanmiglingam and Coulouris 2002].

All these tools – network simulators and context controllers – were mainly developed for evaluation purposes, but they may serve verification purposes as well. Now, test platforms using such tools come with problems that are classical in the framework of distributed systems (e.g., synchronisation of test components), plus additional problems that are more related to the discrete simulation of continuous processes (see e.g., in [Schroth et al 2005], a discussion on the coupling of car traffic and network simulators, so as to mitigate deviations in the node positions and to allow for movement without leaps). Also, the implementation of test experiments is likely to involve the development of specific, low-level scripts. To the best of our knowledge, there is no testing framework or language comparable to what exists for traditional distributed systems.

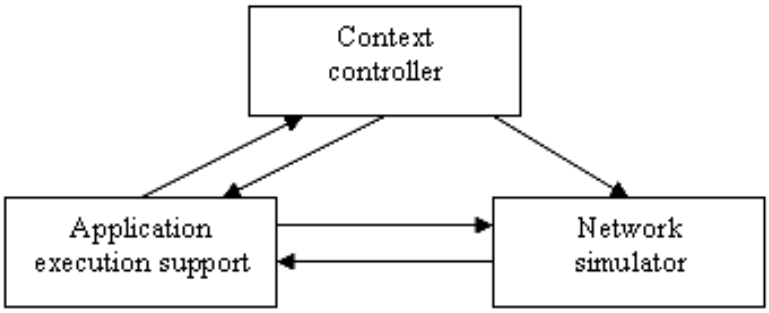


Figure 1: High-level view of the platforms used by [Morla and Davies 2004, Schroth et al 2005]

3.2.2 Modeling issues to support test activities

Model-based test generation techniques have been little explored in the case of mobile computing systems. It seems that the typical practice is to select test cases manually from the informal application requirements, possibly with the aid of classical scenario languages.

Still, first contributions came from the protocol testing community. As said in Section 2, a widely used protocol modeling language is SDL, and there are methods to synthesise test cases from an SDL specification and a set of test purposes. In [Noudem and Viho 2005], the authors study MIPv6, a protocol enabling nodes to remain reachable while moving around in the IPv6 Internet. In [Cavalli et al. 2004], the studied protocol is in the ad hoc domain: the Dynamic Source Routing protocol (DSR) that allows routes to be discovered and maintained in multi-hop wireless ad hoc networks. Both [Noudem and Viho 2005] and [Cavalli et al. 2004] had to tackle the problem of not having mobility-related concepts in SDL. They had to add specific components (one for each node, plus a centralised controller) to capture the notion of communication with neighbors. They also had to

choose a baseline configuration for test synthesis, e.g., in [Cavalli et al. 2004], five nodes connected according to a certain topology, with one link becoming broken after 15 seconds.

These contributions show that it is possible to transfer model-based testing technology to the mobile computing field, at least at the protocol level. This is done using modeling tricks, and predetermined system configurations.

For long-term research, it would be desirable to investigate testing from behavior models already equipped with primitive notions of location and mobility. Some low-level formalisms based on process algebras, like the "Mobile Ambients" [Cardelli and Gordon 2000], offer such notions. Basically, the mobility primitives consist of entering and exiting administrative domains that are hierarchically organised. This view is mostly adequate to express logical mobility (mobile computation), or physical mobility from one infrastructure access point to another. But physical mobility would require further investigation, e.g. to account for dynamic ad-hoc networking. Also, the ambient calculus is more designed for theoretical studies than for specifying systems. There is a need for higher-level formalisms usable by engineers. For example, some UML extensions proposed in [Baumeister et al. 2003, Grassi et al. 2004] both have a stereotyped definition of location that resembles the ambients concept. They differ on how UML notations are impacted. In [Baumeister et al. 2003], the authors modify the syntax of activity and sequence diagrams, allowing them to express system scenarios with both mobility and computational aspects. The work in [Grassi et al. 2004] keeps the notations unchanged, so as to remain compliant with UML 2.0. Potential location configurations are indicated in deployment diagrams and statechart diagrams are used to model the logic of mobility, this logic being kept separated from the application logic. None of these proposed UML extensions has been given a formal semantics.

To conclude, it is not unfair to say that there is currently no well-established modeling framework to support the investigation of test generation approaches for mobile computing systems. This is specifically true for applications in the ad hoc domain, like the GMP case study presented in this deliverable.

4 Case study

This chapter includes the insights gained from the detailed analysis of the GMP protocol mentioned in the beginning of this deliverable. We choose this particular application because it is a good example of a non-trivial, mobile-based service. It addresses a very complex problem, where besides the challenges raised by traditional distributed systems (e.g. atomicity, asynchronous behavior) problems from the mobile environment (e.g. frequent topology changes, network delays) also arise. The protocol has a specification with textual description and pseudo-codes describing the important methods published in [Huang et al. 2004], moreover the authors created a Java language implementation as part of the LIME [Lime] open source middleware for mobile applications. The implementation is not just a small example program: it consists of 4 KLOC of Java code, contains 22 Java classes and after all the components are started there are 6 concurrent threads.

These properties make this application a good candidate for a Verification and Validation (V&V) case study. In the Hidenets project we conducted an analysis by (1) reviewing the specification, (2) creating a UML model for the implementation of the protocol, (3) comparing the specification to the implementation, and (4) testing the implementation. The following sections present the results of the above steps, which have been partly published in a conference paper [Waeselynck et al. 2007].

4.1 Review of the GMP

The review phase included the inspection of the protocol specification based on the description given in the paper [Huang et al. 2004], and the creation of a UML model for analyzing the implementation.

4.1.1 Overview of the protocol

The problem the protocol aims to solve arises in mobile computing environments. Mobile hosts form short-term groups to complete together an action, for which it is essential that all group members have the same consistent view of the group. The main difference compared to the fixed network environments is that in a mobile environment group changes are frequent events, because hosts often move out of the range of each other. Hence, even a short inconsistency is not allowed during the group change operation, which can occur in other protocols.

The other complexity in developing mobile applications is to design appropriate solutions for unrecoverable mobility-induced failures. If two hosts move out from each others range while a message is being sent, the sender does not have a mechanism to check whether the other party received the message. In fixed networks the communication failures tend to be repaired after a time, however, if the two hosts move away from each other in a mobile environment, they will not have the possibility to communicate again. For this reason, the aim of the protocol is to prevent the unannounced leaving of the group by making the hosts report their departing in time.

Since the focus is on mobility-induced disconnections, other classes of threats are explicitly excluded from consideration. It is assumed that the communication service is reliable and that the message delivery time has an upper bound t_d. There are no host crashes, nor omission and performance failures caused by network congestion.

To account for the mobile settings, the novelty of this GMP is the safe distance concept. If two nodes are within this distance, the protocol guarantees that, regardless of their moving pattern, they can finish the task in progress. The leader of each group monitors the location of the members. It splits the group if they start to move away too far from each other, thus preventing unannounced changes. Before the merging of groups, the safe distance criterion is also checked.

4.1.2 Inspecting the specification of the protocol

The first step was the review of the specification paying attention to the following questions: (i) is the specification complete, (ii) are the properties of the specification testable, and (iii) are the worst case scenarios presented by the mobility of nodes covered in the protocol?

4.1.2.1 Properties of the protocol

The membership service is characterised with the properties presented in Table 1 (see Section 2.1 in [Huang et al. 2004] for a more precise definition of the properties).

Table 1: General properties of the protocol

Property	Informal definition
Self inclusion	A node is always a part of its membership view.
Local monotonicity	Group identifiers installed on each node are in increasing order.
Membership agreement	If two nodes have the same group id, then they have the same membership view.
Initial membership view	A node always installs itself as the only member in its view when it starts.
Membership change justification	The successor of group g w.r.t p is either a proper superset or a proper subset of the group g.
Same view message delivery	If node p sends a message m_{pq} to node q at time t, and q is in mem(p, t), then m_{pq} is guaranteed to be delivered to q at time t_0, and mem(q, t_0) = mem(p, t).[a]
Conditional eventual integration	If two groups satisfy the merging criteria and do so for long enough, they will merge into one group.
Conditional group split	A group splits only if it is necessary.

[a] mem(p, t) yields p's local view of the membership at time t.

It was noticed that the definition of *Same View Message Delivery* only implies that the two nodes should have the same member list. It does not require that the group identifiers should be the same. For example, a split and a merge could occur between the sending and receiving of the message.

The formulation of two properties, *Conditional Eventual Integration* and *Conditional Group Split*, was found inadequate from a verification perspective. For example, in the definition of Conditional Eventual Integration, the constant for how long the merging criterion shall be satisfied is not constructively defined. This makes the verification of the protocol with respect to this property not feasible. Also, the requirement does not explicitly specify the case when more than two groups are involved in a merging. Similarly, conformance to *Conditional Group Split* cannot be practically verified.

It is worth noting that the formal specification of such properties would be specifically demanding. Both the merge and split criteria involve some spatio-temporal relationships between the mobile nodes. How to determine the right abstractions for them, and which specification formalism to choose, are open problems.

4.1.2.2 Safe Distance Calculation

The key concept of safe distance is also quite challenging. Intuitively, worst-case scenarios involve the joint consideration of both the movement of nodes and the distributed execution of group-level operations. Two definitions are given for safe distance in [Huang et al. 2004], respectively in Sections 3 and 5. They are reproduced in Table 2. The left one is the general definition of an abstract threshold *r*. The right one refines the general definition to account for the detailed GMP specification, yielding a value d_s.

Table 2: Safe distance calculation

$r \leq R - 2v(t + t')$	$d_s = R - 2V_{max}(t_u + 7t_d)$
R: range of communication t: time of application level task t': time for a group operation	v, V_{max}: maximum speed t_u: location report period t_d: maximum network latency

It is not clear how the two formulas could be mapped. It seems to us, that the calculated result $t_u + 7t_d$, which accounts for the time needed for a split ($2t_d$) taking into account: i) the uncertainty about nodes location (t_u+t_d) and ii) that a merge could be in progress which cannot be aborted ($4t_d$), corresponds to only *t'* from the left formula, and the t part is missing.

The definition and calculation of t_d is also ambiguous. In Section 1 of the paper [Huang et al. 2004] it is referred to as the network delay, while in Section 5 it is defined as the network plus queuing delay. However, calculating the maximum queuing delay in the protocol can be problematic, because messages can block each other, e.g. if a merge is in progress after the group change notification is received, the processing of messages is blocked. As can be seen, calculation of safe distance is a very complex problem and probably needs more attention.

4.1.2.3 General mechanism of the protocol

The functionality of the protocol is divided into two parts: (i) group discovery manages the discovery and reporting of newly arrived hosts, (ii) group reconfiguration performs the merging and splitting of groups when needed. The protocol uses a centralised approach, every group has one leader. The leader collects the location and discovery information of the hosts. Using this information, it checks the group merging and splitting criteria, and starts a group change operation if necessary. The group is uniquely identified by the leader's id and a group change sequence number, which is increased after each group change operation.

The discovery protocol uses the following mechanism. The hosts periodically report their location to their leaders. Apart from that they periodically also broadcast a *hello* message with their host and group id to let other nodes know of their presence. If a host receives a hello message from a host not in his own group, and the new host is in safe distance, it reports it to its leader.

A reconfiguration can be triggered by two actions. On one hand, a leader could receive reports from new nodes from its group members. In this case, it collects the possible merge candidates, the leaders of the other groups, and initiates a merge. A merge is a two-phase operation with request – acknowledge – commit messages. The messages sent during a merge are illustrated on Figure 2.

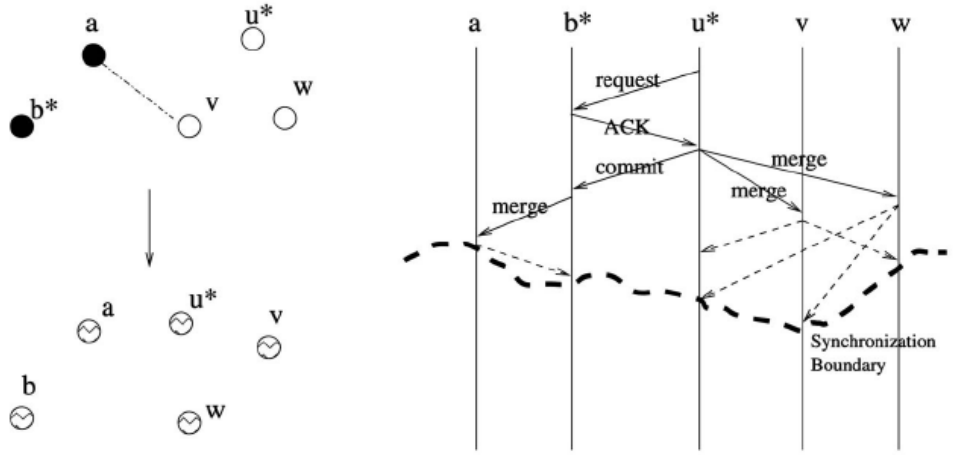


Figure 2: An example for the merging process. On the left side the change of the group membership (star means the leader of a group), on the right side the messages exchanged are depicted.

On the other hand, after a location update a leader can discover that its group is not safe by checking the safe distance criterion. In this case, it starts a split operation and partitions the group into new groups that indeed satisfy the criterion.

The protocol includes well-known mechanisms to counter the typical problems in group membership and group communication situations.

- Atomicity of the operations: the atomicity of the operations is *intended* to be guaranteed by *flush* messages. After receiving a change order, a host shall send flush messages to all its former members, and delay all communication until all the other flush messages arrive.
- Message sequence number: to counter the processing of future messages, all messages are tagged with a group change sequence number. In this way, the processing of future messages can be delayed.

The paper [Huang et al. 2004] includes further details of the protocol. However, because most of the behavior of the protocol is specified in English text format, and the control flow for the pseudo code functions is absent, there are unclear parts in the specification.

4.1.2.4 Clarifications for the specification

The main concerns raised regarding the specification are the followings.

Definition of the Control Flow. The pseudo-code description is very useful to understand the functions of the protocol. However, the flow of control is missing. It is not stated whether the functions may be executed in parallel, and how they may interrupt each other. For example, can a location update be processed during a merge? It is even more problematic that the atomicity of the group change operations is not clearly stated. It is hinted, but it does not appear in the formal description, e.g. like a precondition for split saying that a merge cannot be in progress. This endangers the whole protocol, as such situations could lead to the violation of the properties.

Flush messages. One mechanism intended to make the operations globally atomic is the including of flush messages, i.e. each group change operation is closed by sending to peers a message indicating that this is the last message from the old membership configuration.

The definition of flushing was found incomplete during the review. For example, it is not specified whether the control and hello messages should be blocked during the flushing phase (probably yes).

Messages that arrive after sending the flush messages are blocked until all other flush messages arrive. If a hello message is blocked, then the uncertainty of the location of a node grows even with one t_d, which was probably not taken into account in safe distance calculation. We are back to the problem of worst-case scenarios combining mobility patterns with distributed execution schemes.

4.1.3 Reviewing the implementation of the protocol

The analysis of the specification was followed by reviewing the code of the implementation. This was performed with the double aim of (1) studying conformance with respect to the detailed specification (here, the pseudo-code description), and (2) looking for classical implementation flaws (e.g., thread synchronisation problems).

4.1.3.1 Static structure of the implementation

To obtain the static structure, the Java code was reverse engineered into UML 2.0 class diagrams. This helped us to identify which entities store which part of the state data.

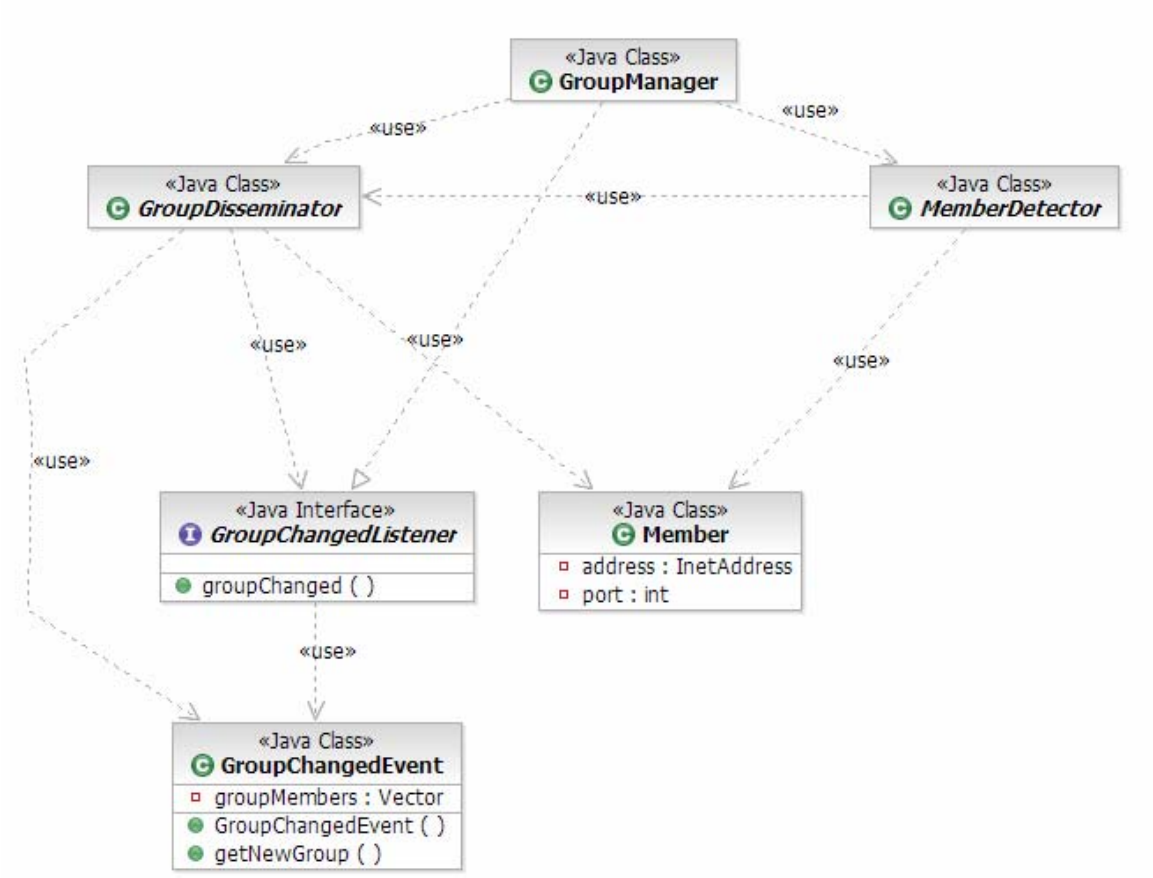


Figure 3: Class diagram of the groupmgmt package of the implementation

The tasks for managing the consistent group view are decomposed into two abstract classes. The *MemberDetector* is responsible for the group discovery protocol, detecting arriving and departing members. The discovery is reported to the *GroupDisseminator*, which decides how this information affects the group, i.e. it can start a merge or a split. The concrete implementation is done in the *SafeDistanceDetector* class, which uses the safe distance criteria for deciding whether a new node

should be reported to the disseminator layer, and the *SinglePhaseDisseminator* class, which, as its name suggests, uses a single phase commit method for the group change operations.

It turned out that the Java code is not the implementation of the pseudo-code included in the specification. We tried to map the message types and the pseudo code functions in the paper to the message classes and Java functions of the implementation. The results are summarised in Table 3, the first column contains the names used in the specification, while the second one uses the names of the Java classes in the implementation. As it can be seen in most cases the one to one mapping is not possible.

Table 3: Mapping the message types

Specification	Implementation
Hello(xy,gid)	~MemberBeacon, but it does not include *gid* and it is not periodical
discovery information	~SPConnectionChanges, but: SPChange includes also loss of connection.
merge-request	– (no explicit request)
ACK(members, sequence number)	~SPGroupInfo, but in the implementation this is sent always by the one who will be a follower after the merge
NACK	– (not present in the implementation, a leader cannot refuse a merge)
merge-commit	– (No specific commit, the other leader also gets an SPGroupChange like other members)
merge-order(new members, new leader, new sequence id) split-order(new leader, new group, new sequence id)	SPGroupChange
Flush-message	– (not implemented)
– (no need, because by sending the gid in hello, others know who my leader is)	SPGetLeader, SPLeaderAddress
– (not in the specification)	SPChangeRefused
– (used internally by the disseminator)	SPChange

4.1.3.2 Dynamic behavior of the implementation

To capture the dynamic behavior, UML sequence diagrams were created manually for the important scenarios. It helped us to analyze how the implementation actually works, what kind of messages are exchanged during the group change operations. The full model with the sequence diagrams showing the behavior of each class is available in [Micskei et al. 2006], here only the most significant differences between the specification and the implementation are shown. The process of a typical merge is depicted on Figure 4.

The protocol uses the assumption, which is by the way not stated anywhere that the detection is symmetrical. When merge-related messages are sent, both leaders are aware of the merge. However, there is no message signaling a request for merge. A scenario was produced where a leader executed a merge upon receiving a SPGroupInfo, however it was not aware of the other group and did not check whether it was in safe distance.

The specification uses a two-phase commit protocol to perform a merge, with a request-acknowledge-commit sequence. In the implementation the changes are performed without the

request and commit phases. After asking about the old membership view, the new leader just notifies the members about the new one.

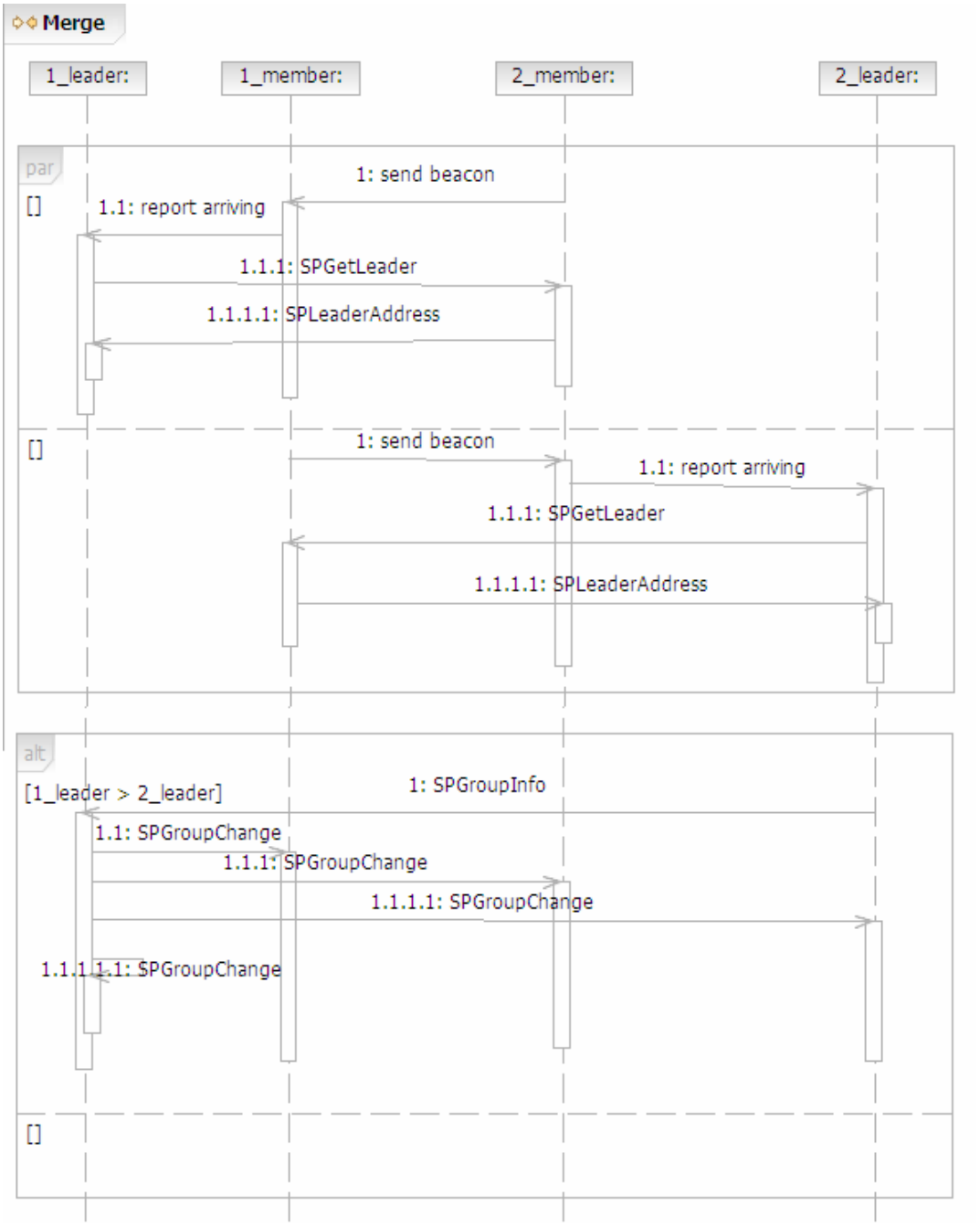


Figure 4. A typical merge

In the specification a centralised approach is used. The group members automatically report new nodes to the leader, and the leader stores every member's location and decides whether the new node is in safe distance. Table 4 contains the data stored by the implementation. It can be seen that the implementation uses a decentralised approach, unlike the specification . Each node monitors the location of its neighbors. Connectivity changes (as opposed to precise location information) are reported to the leader only if needed.

Table 4. Data stored by the implementation classes

Class	Data
Member	INetAddress, port (public)
MemberBeacon	Member, Location object (public)
Detector	Beacons for neighbors = those members, who are connected to this node (private)
Disseminator	Leader as Member object (private) Leader only: logical connectivity graph of the members in the group (private)
GroupManager	Self as a static Member object (public) Group members as list of Member objects (private)

There are also two important features that are not included in the implementation, the *sequence numbers* (to implement group ids) and the *flush messages* (to close group change operations). The lack of sequence numbers resulted in failures shown in the testing section (Section 4.2). As for the flush messages, in Section 4 of [Huang et al. 2004], the authors explain that their implementation separates the application messages and the GMP messages, sent on a different channel. The implementation of mechanisms to guarantee atomicity (e.g. flush or timeout) is then left to each application. In our opinion, the problem cannot be delegated to the application because atomic group changes must be controlled at the level of the GMP package.

The local atomicity of the split is guaranteed in the implementation by blocking the message processing thread with built-in Java synchronisation mechanisms during the coordination of a split. For a merge, however, this processing thread is not blocked.

Finally, the fact that changes are performed without request and commit phases is potentially dangerous. The nodes always accept new configurations sent by anyone (not just their current leader).

4.1.3.3 Scenarios violating the properties

During the review, two scenarios were identified as potentially violating the properties of the protocol. The first one is caused by the atomicity problems mentioned above; the second one is related to global ordering of messages.

Scenario 1: A split can interleave with merge and thus create an inconsistent group view (Figure 5).

Here Nodes 1 and 2 are part of a group led by Node 2, while Node 3 is a leader of another group. Node 2 detects that 3 is in safe distance and queries its leader. It detects that the other one should lead the merge (because its id is higher), so it sends its member list and connectivity information in an SPGroupInfo message. However, before the SPGroupChange containing the new membership view arrives from Node 3, Node 2 detects a departure from its old group, thus performs the split. By the time the SPGroupChange from Node 3 arrives, it contains also the nodes that departed meantime, thus the scenario results in an inconsistent group view which violates the safe distance concept.

Scenario 2: Processing of future messages could violate the same view delivery property (Figure 6).

In this scenario, Nodes 3, 2 and 1 are in one group at the beginning, 3 is the leader. Node 3 detects that it will be separated from 2 and 1, so performs a split, and sends the new group to 2 and 1. But due to network latency, 1 receives the message far later than 2. In the meantime, 2 (who is the leader of the new group), detects also that it will be separated from 1, performs a split, and informs 1 about the group change. Node 1 processes 2's message before 3's, so upon receiving both messages, it thinks it is still in the (2,1) group. The problem is that there is no blocking until all

members apply the group change. Future messages can be processed before completion of the current change.

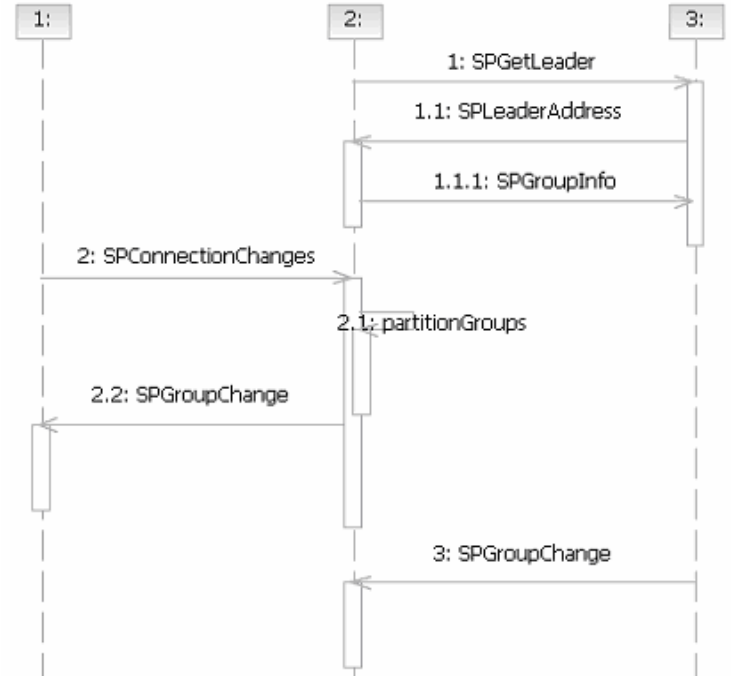


Figure 5. Split interleaving with a merge. Message SPGroupChange contains the new membership view.

As can be seen, reviews can be effective to reveal design flaws and problematic scenarios. The UML models proved a useful support for the analysis. They allowed us to gain deep insight into the code executed by each node. However, reasoning about the global behavior of several moving nodes remained difficult.

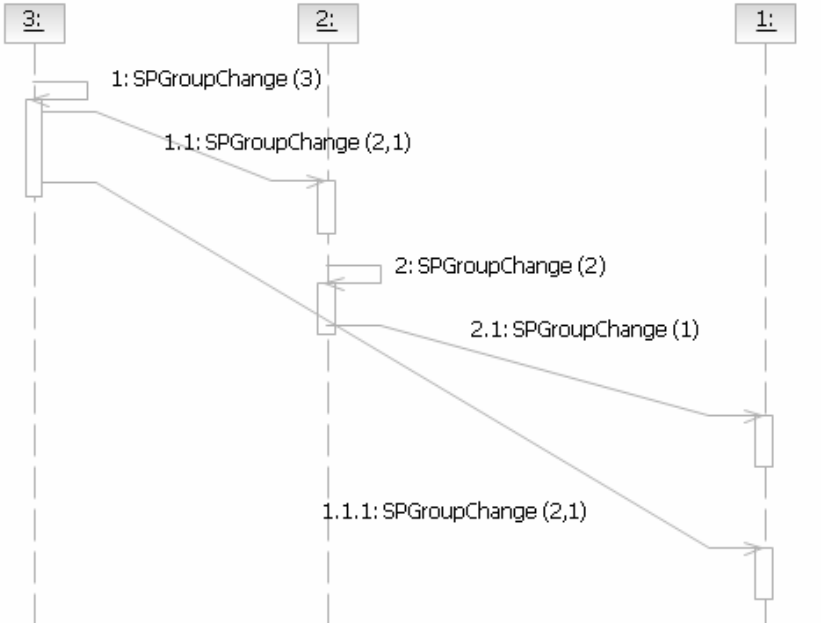


Figure 6: Accepting future messages

4.1.4 Conclusion of the review

The following main issues were found during the review of the specification and the implementation.

- Some of the properties of the protocol are incomplete or not testable.
- The calculation of the safe distance is potentially not accurate taking into account all the possible complex movement patterns and combinations of group change events.
- The English language specification of the protocol is sometimes ambiguous, and the pseudo code definition of the key functions is also not sufficient.
- The atomicity of the protocol is not guaranteed in worst-case scenarios.
- The implementation lacks key features that are essential to the correct behavior of the protocol.

These are just the main concerns, a full list of the identified issues can be found in [Micskei et al. 2006].

The analysis showed general problems that are relevant for any mobile application dealing with mobility and cooperation of hosts. The same analysis (e.g., testable properties, atomicity of operations, identifying unclear parts in the specification and modeling static and dynamic structure) could be performed for many applications developed for the Hidenets specific use cases. Moreover, the case study highlighted the following issues that served as a guideline when defining the testing framework for Hidenets:

- it is not easy to model mobile system instances, without a suitable notation and modeling methodology serious design defects could be introduced,
- the definition of properties containing spatial and temporal information is a complex task, but the correct formulation is essential to the later verification steps.

4.2 Testing the GMP

The test experiments we performed permitted us to familiarise ourselves with mobile computing from a testing point of view. It can be seen as a form of passive testing: the GMP is run using a synthetic workload (random movement of nodes), execution traces are collected, and it is *a posteriori* checked whether the implementation satisfies its high-level requirements listed in Table 1. Since two of the required properties (*Conditional Eventual Integration, Conditional Group Split*) are lacking a precise definition, they are not retained for testing. Two other properties (*Local Monotonicity, Membership Agreement*) depend on the availability of group identifiers, which is an unimplemented feature. Thus, we slightly modified the original source code to introduce this feature. This was made in accordance with the paper specification [Huang et al. 2004]: each group is identified by a pair (leader id, group change sequence number).

We describe below the test platform (§4.2.1) and system parameterization (§4.2.2) used in our experiments. Test results are presented in Subsection 4.2.3. They are followed by a discussion in Subsection 4.2.4.

4.2.1 Test platform

A test platform is needed to run the GMP on a set of mobile nodes and observe the results. As a first step, we decided to build a simple platform based on the prototyping facilities offered in the source code distribution. An overview is given in Figure 7.

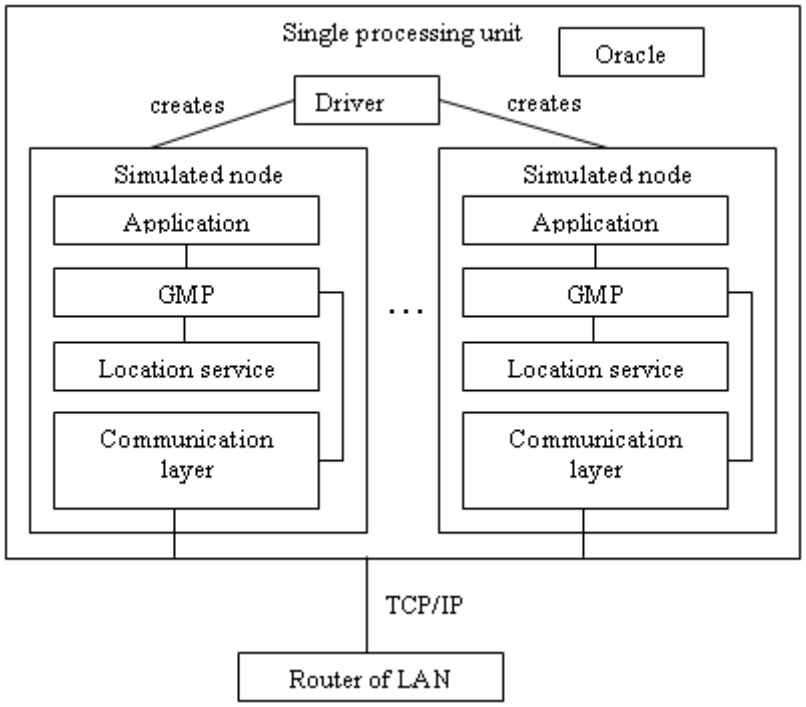


Figure 7: Architecture of the test platform

At the top of the architecture, a test *driver* creates a random number of nodes. All nodes physically reside in the same processing unit, and inter-node communication is actually made via the wired LAN (using TCP/IP). In each node, the *GMP* instance is connected to two stub components: an *application* stub to be notified whenever the group changes, and a *location service* stub delivering physical coordinates for the node. The application stub also generates group traffic at the applicative level. At random instants, it sends its membership view to all peers currently in the group. A receiving application stub may then perform comparison with its own view, and report inconsistency in a log file. The log file also contains the group change notifications, and the dated trace of the messages received or sent by the GMP instance. After completion of a run, a test *oracle* component collects the log files of all nodes for post-mortem analysis. It implements checks for the violation of any of the target properties.

It is worth noting that the properties are tested under the GMP assumptions (e.g., no crash of nodes, reliable communication). Hence the platform does not need to emulate complex real-life problems. But even under such strong assumptions, we are aware that the used prototyping platform has limitations, which will be analyzed in the discussion following the test results (§4.2.4).

4.2.2 Parameters of system configuration

Once the test platform is built, a number of parameters must be instantiated to determine the experimental configuration. These parameters can be classified into two categories:

- Parameters that are dimensioning with respect to the GMP, like the transmission range of nodes.

- Parameters that are more related to the tested workload, like the specific mobility model used to generate (x,y) coordinates for the nodes.

Table 5 provides an overview of the values used in our experiments. Applying the formula for safe distance calculation, the valuation of GMP parameters yields a safe distance of 140 m. As regards the workload parameters, it is worth noting that the GMP functionalities (in terms of split and merge operations) are indirectly activated via the generated node positions.

Table 5: Parameterization of our test experiments

GMP parameters	Value
Transmission range R	300 m
Maximal speed V_{max}	10 m/s
Period for location update t_u	1 s
Network latency t_d	1 s
Workload parameters	**Value**
Number of nodes	Random in [3,15]
Start time of a node	Random in [0, 1000] milliseconds
Initial position	x,y: random in [-150,150] m
Mobility model	Random movement at maximum speed
Applicative messages	Random delay between two messages in [0,10000] milliseconds
Duration of a run	5 minutes

4.2.3 Test results

The experiments involved 100 runs, 82 of which exhibited violation of at least one property. The violations were for:

- Local properties, *Local Monotonicity (LM)* and *Membership Change Justification (MCJ)*.
- Global properties involving several nodes, *Membership Agreement* (MA) and *Same View message Delivery* (SVD*)*.

Table 6: Analysis of a sample of 164 fail scenarios

Violation	**Concurrent splits and merges**	**Concurrent merges**
LM violation only	5	38
MCJ violation only	10	19
MA violation only	10	0
SVD violation only	18	22
2 properties violated	18	16
3 properties violated	6	2
Total number of scenarios	67	97

An example of concurrent split and merge with multiple violations is given in Figure 8. It is an interesting variant of the split & merge scenario found by reviewing the implementation (Figure 5). In the original scenario, a group leader performs a split operation while being explicitly engaged in a merge operation. In the variant of Figure 8, the group leader performing the split (Node 7) is totally unaware of the on-going merge, which is initiated with another member of its group (Node 5). Let us explain the details of the scenario.

Since each 5-minutes run could actually involve many violations, it was not possible to analyze all of them. We extracted a few scenarios from each failing run, by analyzing the first violation and

then by searching later scenarios violating new properties. A total amount of 164 scenarios were extracted. As can be seen in Table 6, the diagnosed problem is always the non-atomicity of the merge operation. A merge operation may interleave with split or with another merge. The problem may induce a variety of failure patterns, ranging from violation of any of the LM, MCJ, MA or SVD property, to multiple violation patterns.

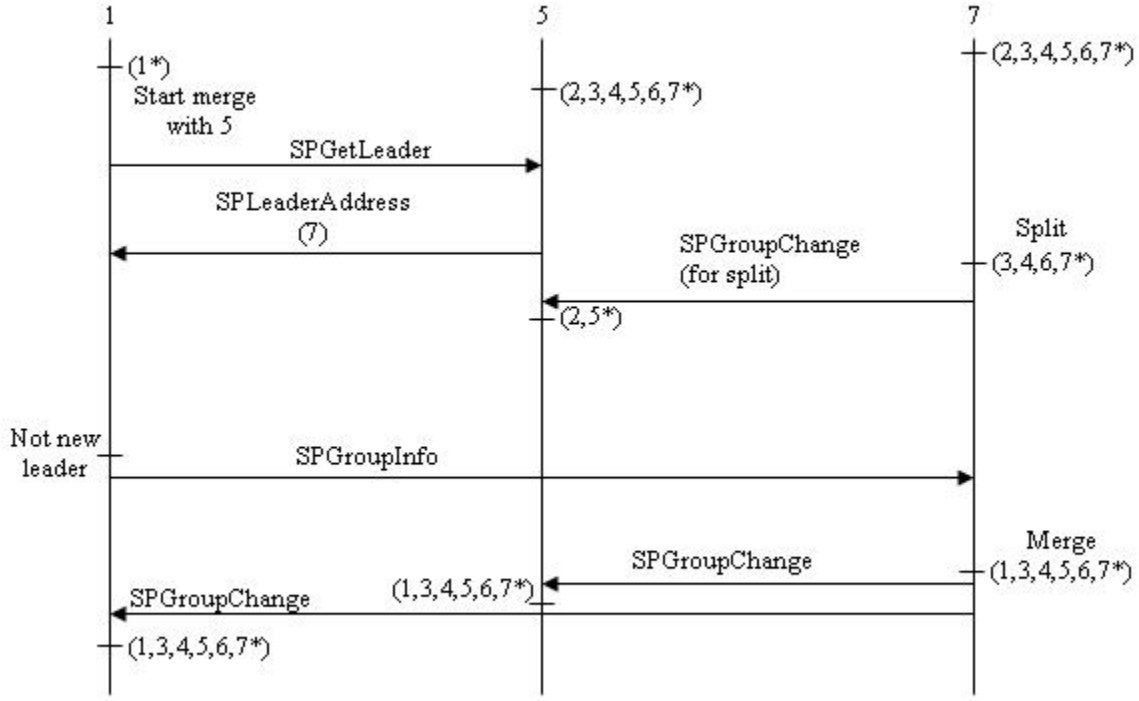


Figure 8: Split interleaving with a merge: a scenario with MCJ+SVD violation.

At the beginning, there are two groups: a singleton with Node 1, and a group with Nodes 2, 3, 4, 5, 6, 7. The second group is led by Node 7, which is noted 7*.

Node 7 detects that its group configuration is no longer safe and performs a split into two subgroups: (2, 5*) and (3, 4, 6, 7*). Then, it sends a series of SPGroupChange messages to notify all members about the split (for the sake of simplicity, Figure 7 only presents the message to Node 5).

In the meantime, Node 1 discovers Node 5 as a new neighbor and starts a merge. It adds Node 5 to its connectivity graph and asks for its leader address (SPGetLeader). When receiving the query, Node 5 is still in the old configuration and replies with its leader address as Node 7 (SPLeaderAddress). Node 1 finds that it will not be the new leader after the merge. It sends its connectivity information to Node 7 (SPGroupInfo). Node 7 must accept the merge, which is made by adding (1, 5) to its membership view. At the end of the merge, the group is (1, 3, 4, 5, 6, 7*). Several problems can be mentioned here:

- Node 5 has been notified of two consecutive group changes: (2, 3, 4, 5, 6, 7*) → (2, 5*) → (1, 3, 4, 5, 6, 7*). Since the last view is not a superset of the previous one, MCJ is violated.
- Node 2 still believes that it is in (2, 5*). An SVD violation is observed via an applicative message.
- More generally, the fact that Node 5 is re-integrated in Node 7's group, while no longer being at a safe distance, raises the problem of the protection offered by the safe distance concept.

Many other variants of split and merge interleaving were observed. In some intriguing cases, we observed a leader sending a series of inconsistent SPGroupChange messages. After analysis, this was related to the multi-threaded behavior of the leader: it started notification about a merge, then switched to notification about a split, and then completed the notification about the merge.

Likewise, concurrent merges generated many scenario variants. In the implementation, there is no mechanism to refuse a merge request, hence making it possible for concurrent merges to interleave in a number of ways.

4.2.4 Discussion

The atomicity of global operations is a classical problem in the field of distributed computing. In [Huang et al. 2004], the authors suggest that each application may choose its own mechanisms for ensuring atomicity. Quoting from them, "Applications with weak consistency requirements may use the Group Membership without any atomicity guarantees". However, our test results show that this is not just a matter of accepting occasional failures: the implementation was found unable to deliver the intended membership service in most cases. Clearly, non-atomicity problems may jeopardise the usefulness of the protocol.

Fixing the implementation would not be a simple exercise. It would require major changes in the definition of the protocol phases, as well as in the synchronisation of threads at the local level. The pseudo-code given in [Huang et al. 2004] would be a good starting point to rework the implementation. But as pointed out in Section 4.1, there are still unclear issues with it, specifically as regards the intra-node control flow and the global mechanism to properly close a group-level operation (flush messages). Whatever the retained solutions, it must be stressed that atomicity must be ensured at the level of the GMP package. The problem cannot be delegated to the applicative level.

After these comments related to the detailed operation of the protocol, it is worth recalling that our aim was not so much to reveal flaws in a research prototype. Rather, it was to tackle a non-trivial example of mobile-based service, and to gain concrete insights into the related testing issues. We comment below on the insights provided by these experiments. This allows us to illustrate some typical concerns that we shall address in the development of a testing methodology for Hidenets-like applications.

4.2.4.1 About the test platform

In the GMP case study, neither a network simulator nor a context controller was used. We simply kept the test platform similar to the prototyping platform delivered by the authors (e.g., wired communication is used). This proved to be a severe limitation.

The test experiments could not produce concrete instances of the second scenario found by reviewing the implementation (Figure 6). This is due to the limitation of the test platform. It does not offer facilities to control communication delays. Hence, it was not possible to trigger the global ordering of messages yielding the target scenario. This is not satisfactory, as global ordering problems are expected to frequently occur in mobile systems. The GMP is intended to accommodate multi-hop communication (a group is safe if any two members are connected via a path along which all consecutive nodes are at a safe distance), and it seems reasonable to assume that communication delays may vary depending of the network topology. This leaves open the possibility for messages to be received in a different order by the nodes (e.g., in Figure 5, Node 1 receives the split from Node 3 *after* the split from Node 2). Such configurations cannot be tested in the current platform.

Generally speaking, the platform does not allow us to control the delivery of messages based on location information. This is a major limitation with respect to the safe distance concept. For example, we would like to observe loss of messages whenever a node moves out of range of its group without having completed a split operation. But in the platform, there is no possibility to tune the communication layer to the current topology. Scenarios impacting the safe distance concept can only be identified by post-mortem analysis (e.g., the scenario in Figure 8 yielding the reintegration of Node 5). A richer test platform would be required to more accurately account for mobility-induced disconnections.

Using a network simulator would have solved these problems. In WP4 [Hidenets D4.1.2] several network simulators were analyzed, this information would be helpful when designing the test platform.

As regards contextual data, the used platform lets each node embed a location service stub (built upon the fake GPS simulators provided in the source code distribution), and hence there is no centralised control of movement patterns. This was also identified as a limitation. First, it was not possible to fine-tune the relative position of nodes, which would have been a serious impediment to active test strategies (directed toward the activation of some specified scenarios). Second, it would not have been possible to simulate complex mobility models, e.g. in physical areas with obstacles.

The GMP case study then provides concrete arguments in favor of a simulation-based test platform that consists of the three categories of components identified in [Morla and Davies 2004]: network simulator, context controller, and application execution support (to interface the application with the context controller and network simulator).

We believe that the identified needs:

- having a centralised control of contextual data

and

- being able to control the delivery of message based on the context,

are representative of fundamental needs when testing mobile computing applications. This case study then confirms that a simulation-based test platform should consist of the three categories of components identified in Figure 1.

4.2.4.2 About scenarios in mobile settings

The graphical representation of fail scenarios proved crucially useful to understand the GMP behavior during testing. We reverse–engineered part of the monitored traces into MSC-like diagrams (see e.g., Figure 8) and this allowed us to gain better understanding of what was happening before property violation. It is worth noting that graphical scenarios were already identified as a privileged support to the preliminary review phase, be it to analyze the discrepancies between the specification and implementation behavior (Section 4.1.3.2) or to identify potential violations of the high-level requirements (Section 4.1.3.3). As regards test result analysis, we obtained complex violation schemes that would have been hardly understandable without a clear picture of how the group operation messages precisely interleave.

However, the classical representation of scenarios in terms of message interleaving was found to lack primitive concepts to account for mobile settings. When trying to interpret the observed behavior, we had to supplement the scenario diagrams by informative statements such as: “Node 7 detects that its group configuration is no longer safe”, or “Node 1 discovers Node 5 as a new neighbor”.

Actually, such statements call for two notions:

- There is an underlying spatial configuration, where nodes are – or are not – at a safe distance.
- The nodes perceive configuration changes by receiving "hello" messages from other nodes at communication range.

These two notions are not explicit in the scenarios obtained from the review or testing phase (Fig. 3, 4, 5, 7). This is so because the spatial topology is not the focus of the scenario languages, and because the notion of broadcast communication with neighbors cannot be conveniently represented. Still, the violation patterns observed during testing depend on which node is the first to send a "hello" message after a configuration change, which nodes receive the message, and how reactions to the locally perceived configuration interleave.

The situation is clearly not satisfactory. Our conclusion is that while scenario representations should play a major role in a V&V framework, classical notations would need extensions to be suited to mobile computing systems.

4.2.4.3 From passive to active test strategies

The test experiments we performed can be seen as a form of passive testing. We did not actively control the activation of specific scenarios. Rather, we ran a synthetic workload where group change operations occur as a consequence of the random movement of nodes. Indeed, the used test platform would not have allowed us to implement an active strategy, for lack of a centralised context controller as explained in Section 4.2.4.1.

Now, suppose that a more adequate platform is used, allowing us to manage the relative position of nodes. From the results of the review phase, which revealed potential atomicity problems, an obvious test strategy would be to try to trigger concurrent split and merge operations. For example, the scenario observed in Figure 8 could be the result of the following test purpose: cover a case where two nodes depart from their old group while simultaneously getting close to another one. This involves managing the trajectory of at least four nodes in such a way that the configurations change occur.

While such scenarios may not be too difficult to trigger by random simulation (indeed, our experiments managed to produce a variety of schemes for concurrent group change operations), the manual tuning of a specific configuration can be quite cumbersome. Note that, in the GMP case study, the underlying constraints are not too complicated (speed is bounded) but some application may involve consideration of more complex mobility constraints. Also, the contextual data are restricted to location data, while in the general case richer contextual data may have to be tuned (think of the distributed black-box example, with a complex policy to choose the neighboring vehicles that will host the back-up).

Our conclusion is that, in the case of mobile computing systems, the implementation of abstract scenarios is expected to be much more difficult than in traditional systems, due to the need to appropriately instantiate contextual data. Automated solutions are needed, possibly with the aid of random simulations using a context controller.

4.3 Summary of insights gained from the case study

The analyzed GMP exhibits problems that are classical in the framework of distributed computing, like the problem of ensuring atomicity of global operations and the one of reaching an agreement on a view. Still, the mobile settings add special challenges to the GMP problem. In more traditional

contexts, the system is likely to alternate between long stability periods and comparatively short instability intervals [Christian and Schmuck 1995]. In mobile settings, a higher dynamicity is expected. Moreover, the group service should be location-aware.

To account for these specificities, the authors have introduced the notion of "safe distance" which is at the core of their GMP. Unfortunately, this core notion proved to be difficult to handle. This is so because the determination of the safe distance depends on the ability to exhibit a worst-case scenario for the GMP. Indeed, the required analysis is quite complex since one has to determine all configurations that may affect the GMP behavior.

Our study shows that rigorous software engineering techniques would have been useful to support the design of such a complex protocol. Many problems were identified without any preliminary execution of the GMP. Analyzing the high-level requirements from the perspective of verification yielded the conclusion that some key properties (conditional eventual integration, conditional group split) were not practically checkable. The review of the design (which was made possible by reverse engineering of the code) showed a lack of traceability between the specification and the implementation. The UML models were a useful support to question design choices. A number of issues were raised, and some potentially dangerous configurations (e.g., a merge interrupted by splits) could already be identified. Then, the test experiments confirmed that the high-level requirements could be violated by the implementation. Although a passive strategy was used, it proved effective to produce many fail scenarios. These concrete scenarios, some of them depending not only on inter-node messages but also on how intra-node threads interleave, would have been difficult to invent based on an intuitive understanding of the GMP behavior.

A general conclusion is that research on mobile-based middleware should have greater concern of V&V issues. Conversely, there is a need for the software engineering community to address the specificities of mobile computing, which raises some open problems:

- Adequate specification and design formalisms still need to be proposed. While classical formalisms (e.g., UML, SDL) may be sufficient to represent one node in isolation, system-level behavior and structure are not easily captured.
- In advanced test approaches, test cases are derived automatically from a model capturing the expected behavior of the application. But considering such approaches might be premature for mobile applications, due to the lack of an established modeling framework. A pragmatic contribution could then be to investigate a language just for expressing scenarios, not for the complete specification of behavior. The user should be able to describe scenarios that are deemed important to be tested, and to implement them on a test platform managing the wireless technology and the mobility of nodes.
- Standard scenario languages do not offer concepts to express the spatio-temporal relationships of nodes as first-class entities, nor do they offer concepts to represent broadcast communication in local vicinity. Hence, they would need extensions to account for the mobile settings.
- The implementation of abstract scenarios raises two issues. The first one is the test platform to be set up. Note that the cost of field testing, as well as controllability and observability constraints, imply that part of the testing activities will have to be performed using emulation/simulation facilities. Fortunately, tools exist that were developed mainly for prototyping and evaluation purposes (like location generators and network simulators) but that can be used for verification purposes as well. The second problem is how to instantiate the scenarios by means of concrete contextual data, for which automated solutions are crucially needed.

Accordingly, our work within Hidenets is directed toward the definition of a scenario-based testing framework that addresses these problems.

5 Towards a scenario-based testing framework

This section first gives an overview of the testing framework that is being developed within Hidenets (§5.1). The framework is based on scenario languages with extensions to better account for mobile settings (§5.2). The usefulness of these extensions is exemplified by scenarios for two case studies, the GMP and the distributed black box (§5.3). We then discuss how to implement some automated treatments of the scenario descriptions (§5.4).

5.1 Overview of the testing framework

As explained in the state-of-the art section, scenario descriptions are widely used to support various test-related activities, such as the representation of requirements, of test purposes, of test cases, or of execution traces. Accordingly, the testing framework depicted on Figure 9 shows scenario-based artifacts that may be produced during different V&V phases.

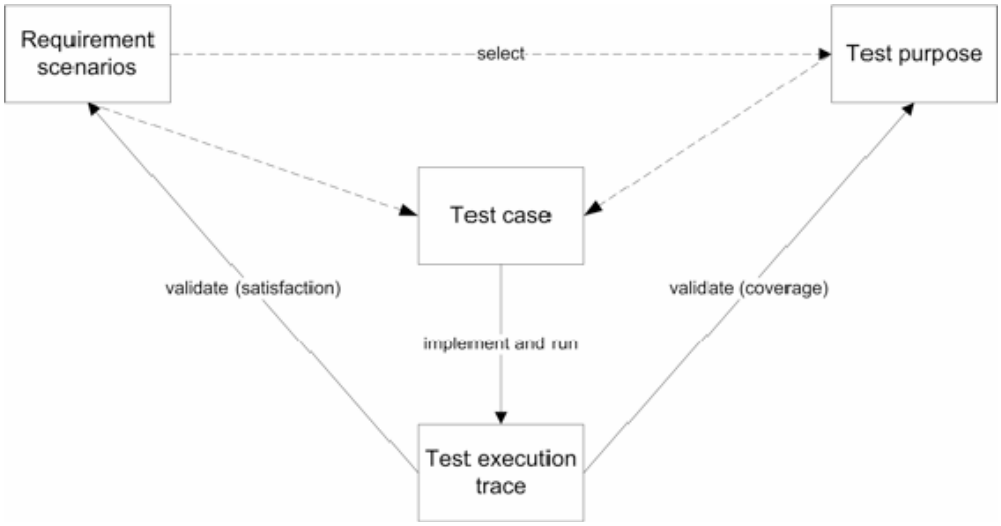


Figure 9: Overview of the testing artifacts

This testing framework does not require commitment to heavyweight formal methods. Hence, the transition from one test specification artifact to the other may be informal, as expressed by the dotted lines. For example, test purposes may be derived informally from the important requirements, and test cases may be proposed by the user to cover some intended purpose. Note that the framework does not *preclude* the use of more formal approaches. Would a complete specification of behavior be available, then the framework could possibly be extended to support formal treatments such as: the verification that the behavior model exhibits the requirement scenarios, or the automated generation of test cases from a model and a set of test purposes. However, such formal treatments will not be investigated within Hidenets.

Even if a complete specification of behavior is not available, some automated treatments become possible by simply using scenario descriptions. They are indicated by solid lines in the figure and will be the focus of our work. The treatments include:

- Checking whether a test execution trace satisfies a requirement scenario.
- Checking whether a test execution trace covers a test purpose.

- Assisting in the implementation of test cases (and more specifically in the production of concrete contextual data).

The key, however, is to have a scenario language with well-defined semantics. Note that each category of testing-related artifact may have a different set of requirements for the language. Some scenarios should be very concrete ones illustrating the exact order of messages, while some of them should allow to capture only a partial view with the important messages only.

Requirement scenarios. Requirement scenarios capture the key properties of the system under test (SUT) that should be satisfied. Obviously the set of requirement scenarios does not yield a complete description of the system's behavior, the scenarios just focus on key properties. In a requirement scenario we would like to express:

- may/must modalities, e.g., to express that a scenario fragment *may* happen, and when it does, then another fragment *must* follow.
- partial traces, i.e. other messages not depicted on the diagrams could be present in valid traces.

Test purpose. The test purpose should indicate what has to be covered during testing and what are the behavior parts of interest. The test purpose should focus on

- specifying behaviors to be covered by test cases,
- should be general in the sense that several possible test cases could be created based on a given test purpose.

Test case. A test case could be created from a general test purpose or directly from a test requirement. It should contain

- only observable and controllable events, not internal events that occur within the SUT
- synchronisation messages between test components
- a precise description of verdict assignment based on the observable events.

Test execution trace. Representing actual, executed test traces requires only a simple language, because it needs to express only one concrete ordering of messages.

As it can be seen from the above the testing artifacts need different levels of detail, which can be achieved by allowing a different set of language elements to be used in them (e.g. allowing different subsets of the UML sequence diagram notation to be used). An ongoing work is to select the appropriate languages elements for each of the above testing artifacts, considering both the desired expressiveness and the ability to provide well-defined semantics.

However, as stated in the previous section, the default elements neither in MSC-like languages nor in UML Sequence Diagrams are suitable to describe scenarios in mobile ad-hoc networks, thus we propose to extend them with the elements described in the next section.

5.2 Proposed extensions to usual scenario languages

The GMP case study allowed us to obtain instructive examples of scenarios combining movement patterns and distributed execution schemes. By trying to represent these scenarios using MSC-like languages, we came to the conclusion that some language extensions would be needed to account for the mobile settings.

Scenario languages typically consider the system behavior in terms of communication events. Some events are ordered while others may interleave in a non-deterministic way, hence yielding a partial order. Figure 10 provides an MSC-like representation of a fail scenario found by testing the GMP. It shows a specific interleaving of split and merge operations. At the end of the scenario, an MCJ violation occurs on Node 5.

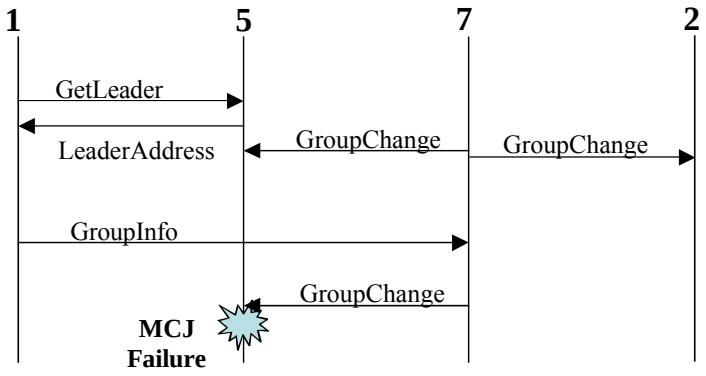


Figure 10: MSC-like view of a fail scenario

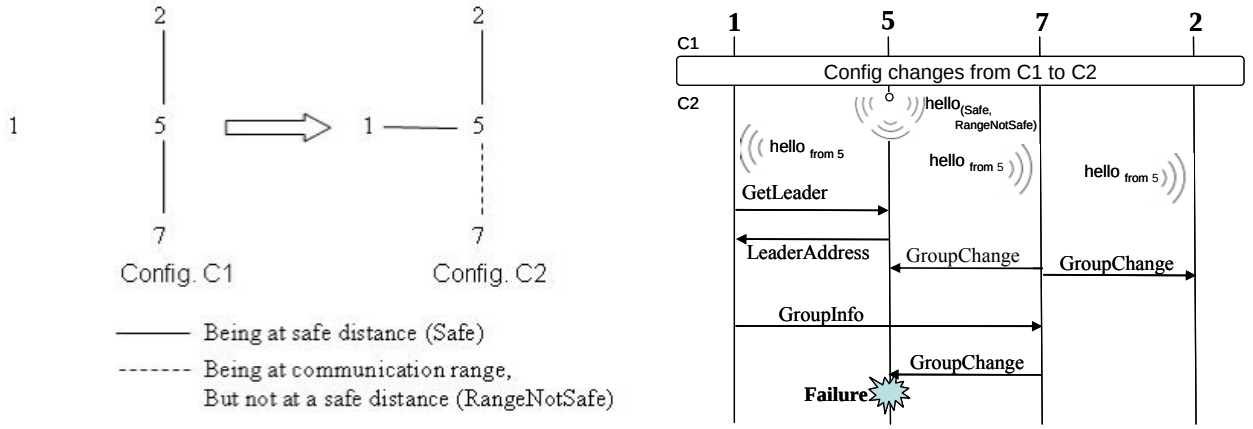
The representation in Figure 10 suffers from several insufficiencies. First, it puts emphasis on the partial order of messages, while the spatial topology of nodes is equally important to characterise the scenario. Second, the MSC notation does not support the expression of broadcast communication – not mentioning broadcast to neighbors. Here, "hello" messages (for group discovery) have not been represented.

After some iteration, we came to the intuitive representation shown in Figure 11. It contains both (1) a graph-like view of the successive spatial configurations, and (2) an MSC-like view of communication events with explicit references to the spatial configurations. It is now clear that the scenario is triggered by the "hello" message broadcasted by Node 5 after a change in spatial configuration. Specifically, Node 5 is no longer at a safe distance from Node 7, while getting close to Node 1. Both the GetLeader message from Node 1 (initiating a merge) and the GroupChange message from Node 7 (performing the split) are causally related to the "hello" message. Without entering the details of the scenario, let us stress the fact that *both* the ordering of messages *and* the topology of nodes are important for the failure to occur. In particular, it is important that:

- the change in spatial configuration breaks the safe path between 7 and 2 (it is precisely the inconsistent treatment of Node 2 in the group change operations led by Node 7 that will induce the MCJ violation on Node 5);
- Node 5 sends its leader address (the leader is Node 7) before being informed about the split. When Node 1 processes this information, the leader address is no longer valid so a fail scenario will appear.

Based on the above, the language extensions to conveniently represent scenarios for mobile computing systems would be the following:

- The spatial relationships of nodes need to be considered as first class concepts. This requires the user to determine the relevant abstraction for such relationships in the target application. For example, two spatial relations govern the GMP behavior: being at a safe distance (which determines the decision to split or merge groups), and being at communication range (which determines the single-hop connectivity). Labeled graphs are adequate to capture such



(a) Evolution of spatial configuration

(b) Partial order of events

Figure 11: Scenario mixing spatial and event views

relations, and we may retain the principle of using a visual formalism to represent the successive spatial configurations.

- Interaction scenarios are decomposed into fragments, where each fragment takes place in one of the previously defined spatial configurations, and configuration changes are represented by a global synchronisation (or equivalently, we have a strong sequential composition of fragments). In this way, causal dependencies involving both configuration changes and node interactions are made explicit. It is also explicit which event occurs in which configuration.
- Broadcast communication is introduced (in Figure 11, this is represented by means of a radio transmission sign). There is a unique send event followed by a set of concurrent receive events. The notation should allow us to specify a subset of receiving nodes, by referring to the spatial relations in the underlying configuration.

We believe that these three extensions would address needs that are recurring when trying to model scenarios in mobile settings. They should be useful whatever the chosen notation variant (e.g., standard MSC, MSC derivatives, or UML sequence diagrams) and whatever the intended purpose of the scenarios (elicitation of requirements, specification of test purposes, of test cases, reverse engineering from execution traces).

5.3 Example scenarios using the extensions

As we stated earlier, the proposed extensions are needed whatever the intended purposes of the scenarios: elicitation of requirements, specification of test purposes or test cases. This will be illustrated in this section by several examples taken from the GMP and the distributed black box application. Moreover, we will try to use different languages to show that they are lacking concepts to express scenarios in mobile settings.

5.3.1 Examples of requirement scenarios

To express the requirements of a system, a language is need where modalities (something *may* or *must* happen) and triggering conditions can be easily expressed. Although UML 2.0 includes many new elements for similar purposes, as discussed by others also [Harel and Maoz 2006], because of

lack of precisely defined semantics UML sequence diagrams may not be the best choice for this purpose. Some MSC derivatives, on the other hand, have proposed well-defined modalities.

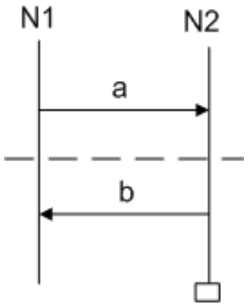


Figure 12: TMSC example

For this reason in this part we will use the *Triggered Message Sequence Chart* (TMSC) [Sengupta and Cleaveland 2006], a graphical framework for capturing scenario-based system requirements of distributed systems. Its significance is the ability to represent requirements that dictate system behavior only when certain *"triggering"* behaviors occur. The syntax of TMSC is based on MSC, it adds two main new features, *conditional scenarios* and *partial scenarios*. Conditional scenarios are represented in the following way. A horizontal dashed line separates the diagram into two sequences: the above part is the trigger and the lower part is the action. If an instance performs the sequence of events constituting its trigger (message *a* in the example illustrated on Figure 12), then the subsequent behavior of the instance must include the sequence of events which constitute its action (message *b* on Figure 12). Partial scenarios are marked by not putting a small bar at the foot of an instance, meaning that the further behavior of that instance is unspecified, it can continue on a separate chart, as illustrated in the case of N1 of the example. Well-defined operators are available to combine multiple charts.

5.3.1.1 Distributed black-box application

As the first example, we want to express the following requirement:

If a car connects to an infrastructure server, and the black-box is requested (from the application) to back up data, it will put its data on the server.

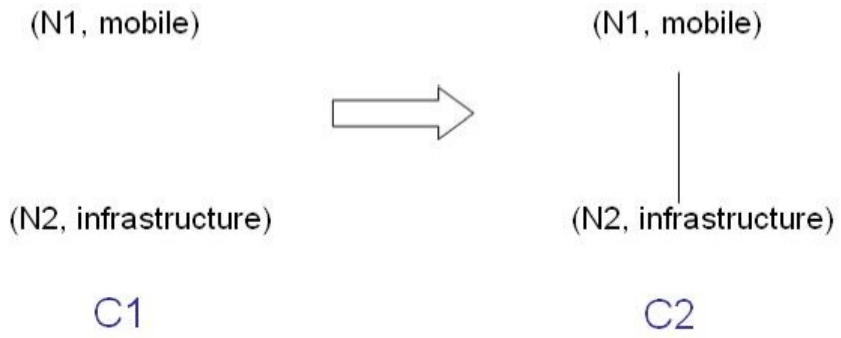


Figure 13: Example: spatial configuration in the distributed black box application

Obviously, the current scenario languages used to express requirements cannot express the configuration change in the network topology that is the triggering condition of the scenario. So we have to find a way to express that a car was not connected to a server (C1), then it became connected (C2).

Our proposed extension on the spatial relationship could cover this situation. Suppose that we have a notion of the configuration change in TMSC, we can illustrate the requirement as in Figure 14.

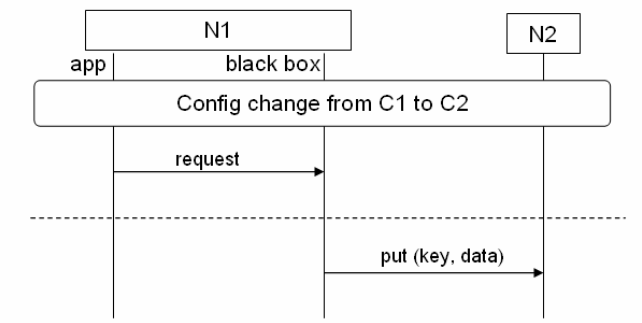


Figure 14: Example: TMSC for the requirement in the distributed black box application

The dashed horizontal line divides the diagram into two parts: the *conditional scenarios* and its *action*s. The former shows the notion of *trigger*, meaning in any system execution, if an instance performs the sequence of events constituting its trigger, then the action must occur. In this case, the spatial relationship is put into the conditional scenario.

None of the instances in the chart have the completion bar at the bottom of the chart, hence illustrating that this is only part of the behavior and not a complete system description.

5.3.1.2 GMP

The informal requirement to describe with a scenario is the following:

If a new node gets into safe distance range to a node, then it will report this to its leader.

The configuration change that triggers the requirement can be described as follows. Two nodes, N1 and N2 are in communication range but not in safe distance range, thus they are not in the same group. Later they move into safe distance range.

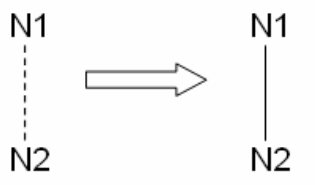


Figure 15: Example: configuration change in the GMP

Using this configuration change the requirement can be easily captured by the following scenario.

In this example, in addition to the spatial relationship, the elements of TMSC cannot express the notion of broadcasting the “hello” message, but it is essential to capture that the hello message is not a direct message to N2 node, instead it is sent to every nearby node. With the broadcast notation defined in the earlier section this can also be illustrated.

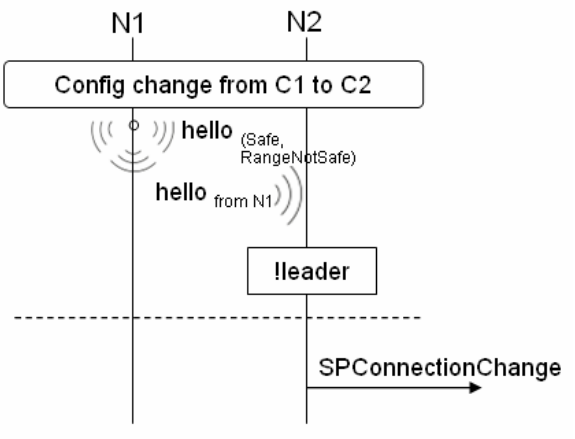


Figure 16: Example: TMSC for the requirement in the GMP

5.3.2 Examples of test purposes

For the test purposes the focus is on specifying only partial scenarios, capturing the behavior that later should be covered in the test cases. In the following examples UML 2.0 Sequence diagrams will be used. The 2.0 version of UML introduced a major change in the sequence diagrams, many new elements were imported from other scenario languages like MSC. Several operators were introduced to combine diagram fragments, like *parallel* or *alternate. Ignore* and *consider* operators were added to express that the diagram shows only a subset of messages that can occur in a valid trace. The following subsections give examples how these elements can be used for specifying test purposes.

5.3.2.1 Distributed black-box application

We want to express the following test purpose.

The case should be covered when a car connects to an infrastructure node, and it puts its data on the server.

Here, the configuration change is the same as described in Figure 13. UML sequence diagrams also lack the notation to express the spatial relations, thus we add the same notation for configuration change.

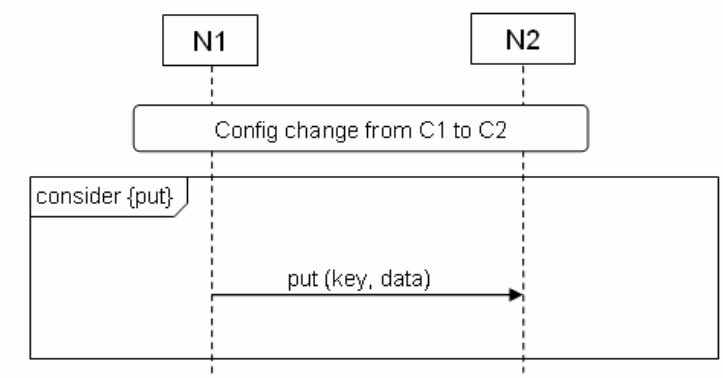


Figure 17: Example: sequence diagram for the test purpose in the distributed black box

The *consider* operator is used to express that in the given fragment only the put messages are depicted. Other types of messages could be interleaving.

5.3.2.2 GMP

In this scenario, we want to cover a concurrent merge in GMP.

Test a concurrent merge scenario, e.g. when a node is performing merges with two distinct groups.

The related spatial topology is the following. At the beginning, we have three singletons N1, N2 and N3, which are in communication range. As the nodes move, the configuration changes: N1 and N3 get into safe distance range to N2. This change is shown in the Figure below.

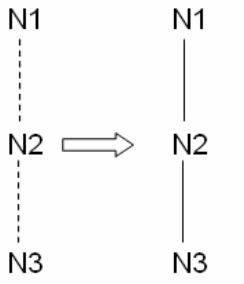


Figure 18: Example: Spatial configuration change in the GMP

Using the configuration changes and broadcasting notation the test purpose can be captured with the following scenario (Figure 19).

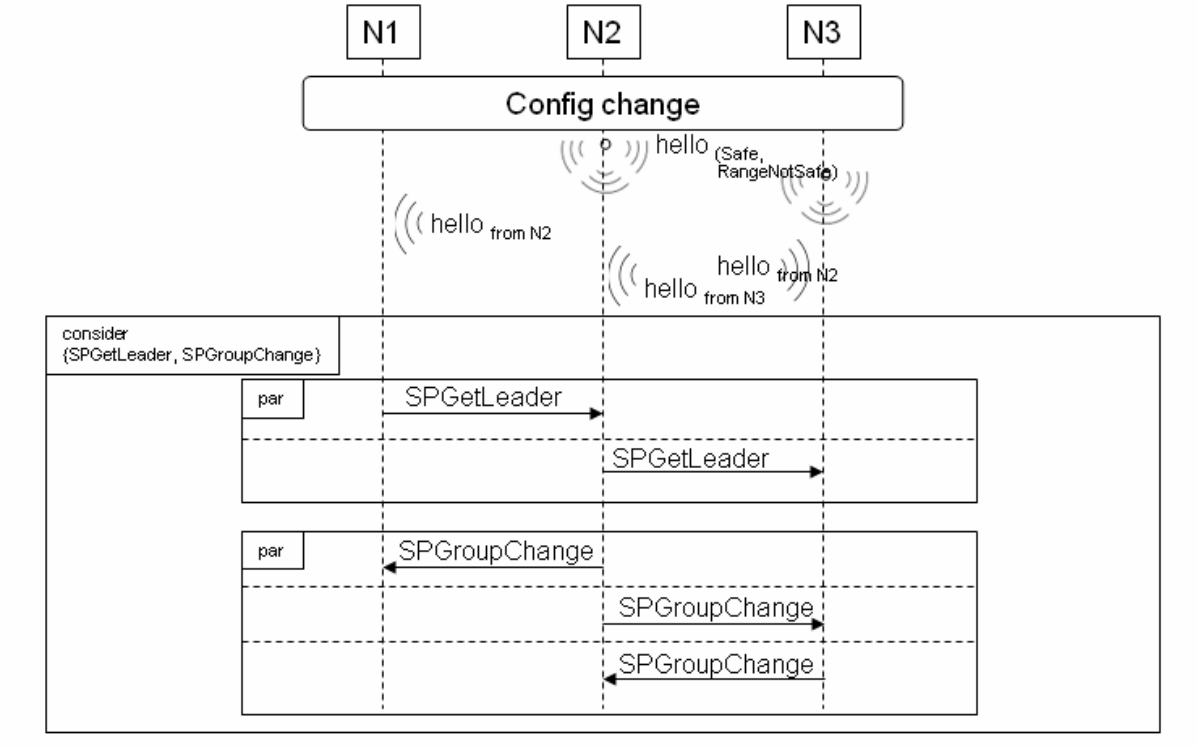


Figure 19: Example: Sequence diagram for the test purpose in the GMP

In this example the partial order of the events can be further specified. As the merge consists of the exchange of many messages, there will be many possibilities of message interleaving. We may then refine the previous test purpose as follows: we want to cover the above concurrent merge, *and* N2 sends its group information to N3 after finishing the merge with N1.

The traces covering this purpose can be expressed with the scenario in Figure 20.

Intuitively, the concurrent merge in this case does not pose problems, since N2 sends its group information to N3 after it established a stable configuration with N1. Indeed, our test experiments showed that this case does not provoke violation of properties. However, other interleaving of messages may cause the violation of properties.

These examples show that it may be convenient, for test purposes, to have language constructs that offer some flexibility to define the partial order of interest (like here the *par* construct).

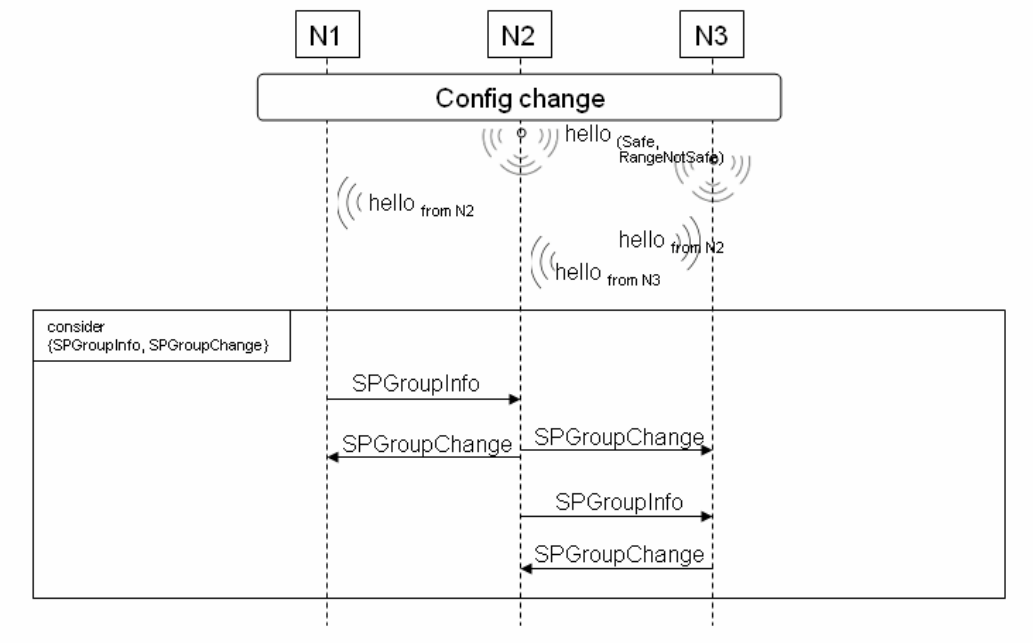


Figure 20: Example: Sequence diagram for the test purpose in the GMP

5.3.3 Examples of test cases

In the test cases we have the additional tasks to identify the role of each component in the scenario and explicitly assign a verdict to the outcome of the test. The UML 2.0 Testing Profile (U2TP) [OMG 2005] was designed especially for this purpose. It defines stereotypes to construct and document the artifacts of test systems. The concepts of the profile are grouped into the following groups: test architecture, test behavior, test data and time. The profile contains well-known testing concepts (like SUT, timer, verdict and arbiter). The usefulness of U2TP is that it gives a standardised notation to express these elements. For a more detailed description of the profile refer to [Dai 2005].

The test case we would like to specify with a scenario is the following.

*Verify that a merge will be done in 4*td between two singletons.*

With this objective, the following test case could be drawn. The configuration change is the same as depicted on Figure 15, two nodes get into safe distance range.

In this example, the GMP is considered as a black-box, the GMP messages are considered as SUT internal, hence they are not illustrated. Therefore, the problem of broadcasting and complex message interleaving is avoided, the testing should only consider the observable actions of the SUT, in this case the groupChangeEvent messages sent to the application. A test coordinator is further

introduced to collect the partial verdicts from the test components and assign the final verdict to the test, in this test case by checking also the time needed to process the merge operation.

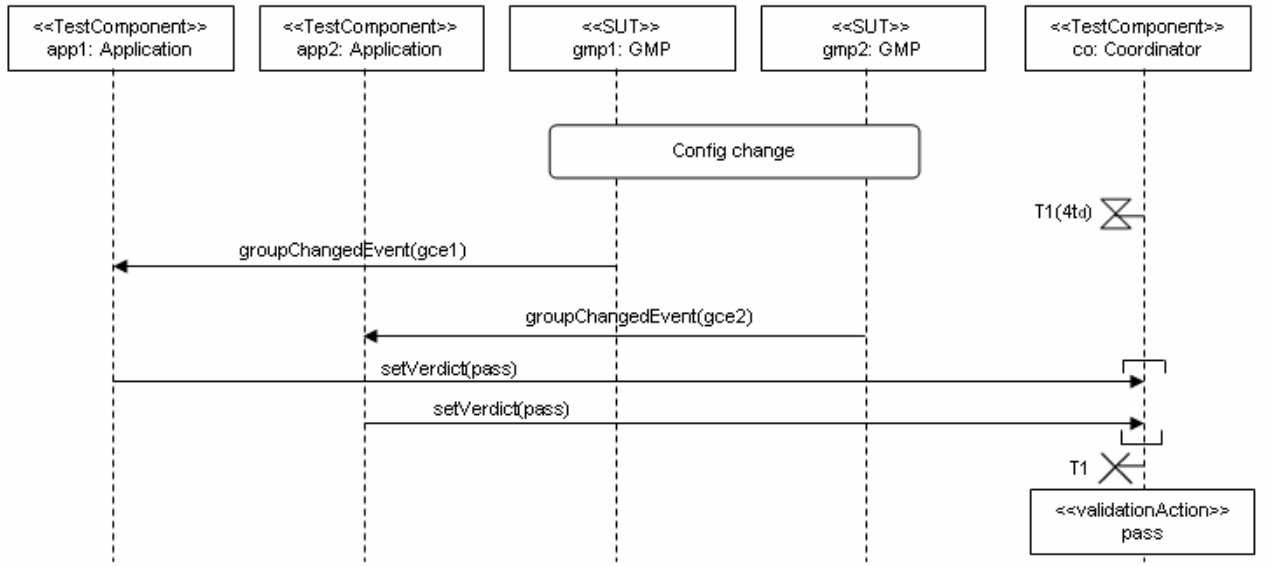


Figure 21: Sequence diagram for the merge test case in the GMP

From these examples we can come to the conclusion that (i) the recommended extensions are useful to express the specificities of mobile ad-hoc environments that were missing in previous notations, (ii) the different types of testing artifacts have different requirements that should be taken into account in the testing framework.

5.4 Analysis of test traces, test context data generation

In the proposed testing framework, scenarios play a central role and may serve several purposes. The associated treatments include:

- Analysis of an execution trace to determine whether a requirement has been violated.
- Analysis of an execution trace to determine whether a test purpose has been covered.

The execution trace, requirement and test purpose are all described by means of scenarios considering the spatial relationships of nodes as first class concepts. In this section, we argue that graph algorithms, and more specifically algorithms for graph matching problems, are relevant to support automation of the above treatments. Such algorithms may even be useful to assist in the production of contextual data that instantiate a test case, as will be explained shortly.

5.4.1 Principle

Suppose a GMP test case has been specified as exemplified in the previous section. In the test case description, movement is abstracted by graph modifications. But ultimately the tested implementation will need to be fed with GPS input data for each node. How to produce concrete contextual data that instantiate the desired evolution of topology, is an acute problem.

Generally speaking, the data may have to obey a complex mobility model (for example, cars moving in a geographical area, with a given network of roads) so that manual production is not realistic. This is why a context controller is needed in the experimental platform (Sections 3.2.1 and 4.2.4.1).

Although we may not be able to specifically build adequate data, we may use random simulation with the hope that the desired configurations will eventually be produced. Our proposal is thus to have a preliminary production phase of contextual data, based on runs of the context controller taken in isolation:

- A run may involve a large number of nodes moving according to the implemented mobility model. At each simulation step, relevant contextual data for each node are recorded.
- The concrete simulation trace is then abstracted by a series of labeled graphs representing the evolution of the system configuration.
- It is then searched whether sub-graphs can match the desired evolution pattern (defined in the test case scenario).
- The list of matches provides alternative baseline configurations for the implementation of the test case.

For example, from a simulation run, we may identify four nodes exhibiting the spatial configurations shown in Figure 11. The recorded contextual data for these nodes (e.g., their location coordinates) can then be extracted from the complete trace, and replayed for the test case execution.

Note that real field datasets, rather than simulation ones, could as well be used as an input to the search for matches, provided such datasets are available. From this perspective, initiatives such as the Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD)[1], that aims at providing a wireless network data resource to the community, may be of highest interest.

Graph matching algorithms may also be used to compare a test execution trace to a test purpose (to detect coverage), or to an application requirement (to detect violation). In both cases, the comparison involves two steps:

- Determine which physical nodes can play the role of the nodes appearing in the specified configuration graphs, according to the observed topology;
- Analyze the order of events in the identified configurations.

Obviously, the first step is typical from a graph matching problem.

We now get into the technical aspects of the search for matches, which calls for building *graph homomorphisms*.

5.4.2 Graph homomorphisms

Let L_V and L_E denote sets of labels for vertices and edges, and let $G = (V, E, \mu, \nu)$ denote a graph structure, where:

- V is the set of vertices,
- $E \subseteq V \times V$ is the set of edges,
- $\lambda : V \rightarrow L_V$ is a function assigning labels to the vertices,
- $\mu : E \rightarrow L_E$ is a function assigning labels to the edges.

Definition 1. Let $G_1 = (V_1, E_1, \lambda_1, \mu_1)$ and $G_2 = (V_2, E_2, \lambda_2, \mu_2)$ be two graphs. A function $f : V_1 \rightarrow V_2$ is a graph homomorphism from G_1 to G_2 if and only if:

[1] http://crawdad.cs.dartmouth.edu/index.php

1. It is injective.
2. $\lambda_1(v_1) = \lambda_2(f(v_1))$ for all $v_1 \in V_1$.
3. For any edge $e_1 = (v_{1s}, v_{1e}) \in E_1$, there exists an edge $e_2 = (f(v_{1s}), f(v_{1e}))$ such that $\mu_1(e_1) = \mu_2(e_2)$.

Intuitively, the notion of graph homomorphism captures the idea of G1 being matched by a subgraph of G2. In the sequel, G1 will be called the *model graph*, and G2 the *host graph*. In the proposed testing framework, the model graph is expected to come from a scenario description, while the host graph is extracted from an execution or simulation trace.

The basic definitions of graph structure and graph homomorphism need to be slightly extended to fulfil our needs.

First, it may be convenient to assign tuple of labels to vertices and edges in order to allow a richer representation of nodes and relations between nodes. For example, in the distributed black box example, a node could be characterised by a 2-tuple <*id, type*>, where *id* would be an integer value uniquely identifying the physical node, and *type* would be an element of *{"Mobile", "Infrastructure"}* that differentiates the mobile and infrastructure nodes. We assume that the model and host graphs have the same dimension of labels, say a *d*-tuple for vertices and an *e*-tuple for edges.

Second, and more importantly, we need to allow label variables in the model graph. In a scenario description, a node may be assigned labels <*n1, "Mobile"*> and it should be possible to detect a matching by a physical node <*100, "Mobile"*> with substitution *n1:= 100*. As can be seen in this example, introducing variables means that the graph homomorphism building has to exhibit a valuation consisting of pairs $(var\_name_i, value_i)$ that unify the labels.

Definition 2. Two labels l_1 from the model graph and l_2 from the host graph can be unified if and only if:

1. l_1 and l_2 are constant values and $l_1 = l_2$.
2. l_1 is a variable name and l_2 is a constant value.

In the second case, the unification is done by a partial valuation $\{(l_1, l_2)\}$.

The unification of two vertices (resp. edges) from the model graph and the host graph calls for the unification of all labels from the *d*-tuple (resp. the *e*-tuple). Care must be taken that the union of the produced partial valuations is consistent. For example, a vertex <*x,1,x*> cannot be unified with <*1,1,2*>, because it would require assigning both 1 and 2 to variable *x*. Similarly, if two distinct vertices (resp. edges) from the model graph have the same variable name appearing in their *d*-tuple (resp. *e*-tuple) of labels, there should be a consistent valuation for this variable.

Definition 3. A valuation *Val* is consistent if and only if there is no two pairs $(var\_name_1, value_1), (var\_name_2, value_2) \in Val$ such that $var\_name_1 = var\_name_2$ and $value_1 \neq value_2$.

The definition of graph homomorphisms may then be modified as follows. In the graph structures, λ (resp. μ) now denotes a function assigning a *d*-tuple (resp. *e*-tuple) of labels to vertices (resp. edges).

Definition 4. Let $G_1 = (V_1, E_1, \lambda_1, \mu_1)$ and $G_2 = (V_2, E_2, \lambda_2, \mu_2)$ be two graphs. A graph homomorphism from G_1 to G_2 is characterised by a pair (f, Val) of a function $f: V_1 \rightarrow V_2$ and valuation *Val* such that:

1. f is injective.
2. For all $v_1 \in E_1$, each label from the d-tuple $\lambda_1(v_1)$ can be unified with its counterpart in $\lambda_2(f(v_1))$.
3. For any edge $e_1 = (v_{1s}, v_{1e}) \in E_1$, there exists an edge $e_2 = (f(v_{1s}), f(v_{1e}))$ such that each label from the e-tuple $\mu_1(e_1)$ can be unified with its counterpart in $\mu_2(e_2)$.
4. The union *Val* of all partial valuations is consistent.

We retain this definition for detecting whether a scenario configuration is matched by a concrete execution trace.

The problem of graph homomorphism building has been extensively studied in the literature. It is thus possible for us to use an existing tool as the basis for the comparison of scenario descriptions and concrete traces.

One of the existing tools has been developed by colleagues from LAAS [Guennoun 2006] in the framework of research on dynamically reconfigurable architectures. The tool searches for the set of all homomorphisms from a model graph G_m to a host graph G_h. It offers the following facilities:

- Vertices may be assigned at most 3 labels, yielding a 3-tuple of type INT × STRING × STRING.
- Edges have at most one label of type INT.
- Label variables are supported for vertices only.

The complexity of the search is polynomial in the number of vertices of G_h, but exponential in the number of vertices of G_m (which is not surprising, since the search problem is known to be NP-complete).

In addition to graph homomorphism building, the API also offers some convenient graph rewriting facilities, both grammar-based facilities, and more basic ones such as a *VALUATE_VERTEX* function. This function takes as inputs a graph and a valuation, and rewrites all vertices according to the valuation. For example, suppose the graph contains a vertex with labels *<x, "Mobile", y>*, and the valuation includes *(x, 2)*. The vertex labels will become *<2, "Mobile", y>*.

5.4.3 Algorithms for analyzing a simulation run

We have retrieved the tool and made preliminary experiments to investigate its functionalities. We are now implementing algorithms on top of it.

The on-going implementation concerns the detection of a sequence of configurations. Suppose a scenario description involves two successive configurations, denoted G_{m1} and G_{m2}. Suppose also that a simulation run is abstracted by a sequence of graphs $G_{h1}, ..., G_{hn}$, where n is the number of simulation steps in the run. We require that the first label in the 3-tuple represents an *id* that uniquely identifies the corresponding node, so as to be able to trace this node from one simulation step to the other. A configuration change from G_{m1} to G_{m2} is then detected when there is an i, $1 \leq i < n$, such that:

- there is a homomorphism *(f, Val)* from G_{m1} to G_{hi} and,
- there is a homomorphism *(f', Val')* from *VALUATE_VERTEX* (G_{m2}, *Val)* to G_{hi+1}.

Note that it is necessary, at the second step, to retain the valuation choices made at the first step. Hence, the model graph is not G_{m2} but *VALUATE_VERTEX* (G_{m2}, *Val)*.

Some additional processing is required to account for nodes that are explicitly created or eliminated by the configuration change:

- G_{hi} should not include vertices with the same *id* as the new nodes in the second configuration.
- G_{hi+1} should not include vertices with the same *id* as the eliminated nodes in the first configuration.

In addition to detecting the configuration change, we also need to identify the temporal window for the two configurations. The reason is that when we will analyze the event view, we need to be sure that a given trace event occurs in the expected configuration. Hence, we have to determine the start date $1 \leq s \leq i$ of the first configuration, and the end date $i+1 \leq e \leq n$ of the second one, again using graph homomorphisms with appropriate valuations.

An obvious problem with this set of on-going developments is that we now have to consider all possible matches for the n simulation steps, which adds to the combinatorial explosion problem. We are currently studying how to alleviate the problem, by taking advantage of the fact that successive graphs differ only slightly from one step to the other. Also, we may consider a bounded number of matches rather than all of them.

Finally, we are planning to investigate algorithms for accommodating richer descriptions of the node configurations in the scenarios, such as:

- Introduction of min, max duration constraints for the configurations. For example, a scenario may require that a given configuration lasts at least k steps, so that the application may reach a stable state in this configuration.
- Introduction of negative fragments of configurations. For example, the scenario may require that the concrete configuration matches G_{mi}, but not $G_{mi} \cup G_{neg}$.

6 Conclusion and perspectives

This Deliverable has laid the basis for the testing framework that is being developed within Hidenets. The goal is to tackle mobile-based applications in the ad-hoc domain, such as the one exemplified by the Hidenets use cases.

The framework is based on graphical scenario languages with extensions to better account for mobile settings:

- The spatial relationships of nodes are now considered as first class concepts, and introduced in labeled graph representations.
- The event view makes it explicit which communication event occurs in which spatial configuration, and configuration changes are introduced as global events.
- Broadcast communication in local vicinity is introduced by means of special symbols.

We have shown that these extensions are useful whatever the original language (e.g., TMSC, UML sequence diagrams, U2TP) and whatever the intended role of the scenario (e.g., to represent a requirement, a test purpose, a test case, an execution trace).

We are now in the process of selecting the appropriate set of language elements for the various testing artifacts. For example, we have identified that the representation of requirements needs modalities, and that constructs to represent partial orders (like the parallel construct) are convenient for test purposes. When the language is stabilised, we intend to investigate the possibility of defining a UML profile for this language, in close relationship with Task 5.1. A difficulty we are faced with is that not all available constructs have clean semantics. For example, UML 2.0 has introduced many new elements in the sequence diagrams, but they come with intriguing semantics problems (see e.g., [Pickin and Jézéquel 2004, Harel and Maoz 2006]). Obviously, it is necessary that the selected language is given a well-defined meaning, so that formal treatments become possible.

The treatments attached to a scenario description will include:

- Checking whether a test execution trace satisfies a requirement scenario.
- Checking whether a test execution trace covers a test purpose.
- Assisting in the implementation of test cases (and more specifically in the production of concrete contextual data).

We have shown that all these treatments involve graph matching problems, at least for some parts:

- The first two treatments ask to determine which physical nodes (from the execution trace) can play the role of the nodes appearing in the specified configuration graphs (from the requirement scenario or test purpose).
- For the third one, it is proposed to extract contextual data from simulation runs, by identifying subsets of nodes that mach the desired configuration.

The graph matching problems can be formalized as a series of searches for graph homomorphisms. We have started the development of a tool, based on an existing implementation of graph homomorphism building. The ability of the tool to handle large simulation runs (when the

production of concrete contextual data is the target) is still to be investigated, possibly at the expense of some optimisations.

In addition to graph algorithms, the first two treatments involve comparing the partial order of events in the test trace and the specified scenario. Automated support for this has been delayed until the target scenario language is stabilised and assigned a well-defined semantics.

We will keep on using two example applications, the distributed black box application and the group membership protocol, to illustrate the concepts developed by this work.