Mobile Systems from a Validation Perspective: a Case Study

Hélène Waeselynck*, Zoltán Micskei°, Minh Duc Nguyen*, Nicolas Rivière*
*LAAS-CNRS
7 avenue Colonel Roche
31077 Toulouse, France
° Budapest University of Technology and Economics, Muegyetem rkp. 3, 1111 Budapest, Hungary
waeselyn@laas.fr, micskeiz@mit.bme.hu, mdnguyen@laas.fr, nriviere@laas.fr

Abstract

Advances in wireless networking have yielded the development of mobile applications. However, sound technology to specify, design and validate such applications is still to be investigated. In order to exemplify some of the challenges that are raised, this paper reports on a case study: a group membership protocol for ad hoc networks. The protocol has been analyzed by reviewing the specification and the code, and then by testing the implementation. The outcomes provides us with hints for research direction.

1. Introduction

Advances in wireless networking technology have yielded the emergence of the mobile computing paradigm. Mobile computing calls for the use of devices (handset, PDA, laptop…) that move within some physical areas, while being connected to networks by means of wireless links (Blue-tooth, IEEE 802.11, GPRS…). Mobile-based applications and services may involve device-to-device or device-to-infrastructure communication. They have to be aware of, and adapt to, contextual changes.

A substantial amount of research is devoted to the development of this recent technology, but correctness issues have been paid insufficient attention so far. There is an urgent need for proper methods to specify, design, and validate mobile computing systems.

To support this claim, this paper reports on a case study. We analyzed a Group Membership Protocol (GMP) for ad hoc networks [1], found in the literature on middleware for mobility. It is worth noting that the authors address a very important problem. A GMP is a fundamental service lying at the heart of distributed, fault-tolerant systems. Its aim is to maintain a consistent view of who is in the group, in spite of the faults that may affect some nodes (faulty nodes are excluded from the group). The problem has been extensively studied for traditional distributed systems, but needs to be revisited to account for mobile settings. In [1], the authors proposed a location-aware protocol that accommodates the disconnections induced by mobility. An open-source implementation of the protocol is given in the LIME middleware [2].

Our analysis of the GMP involved several steps:

- informal review of the paper specification,
- reverse engineering of the source code to produce UML models and analyze the design,
- preliminary test experiments according to a crude strategy (nodes move at random).

They showed that several requirements are violated by the implementation. But the aim was not so much to reveal flaws in a research prototype given as a proof-of-concept illustration. Rather, it was to tackle a non-trivial example of mobile-based service, and to gain concrete insights into the related validation issues. We experienced that the consideration for mobility may yield complex behavior patterns, making rigorous V&V methods a major concern. The case study exemplifies some of the challenges to be addressed, and provides us with hints for research direction.

Section 2 of the paper introduces the studied GMP. Sections 3 to 5 reports on the correctness issues raised at each step of the study (review of the specification, of the design, and testing). Section 6 discusses related work. Section 7 concludes on research perspectives.

2. Overview of the Case Study 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. Every group has one leader. The group is uniquely identified by the leader's id and a group change sequence number, which is increased after each group change operation.

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.

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. The membership service is then characterized with the properties presented in Table 1 (see Section 2.1 in [1] for a more precise definition of the properties).

3. Review of the Specification

Specification reviews typically aim at revealing inconsistencies, omissions, or ambiguities in the specification. In our case, the specification document is the material published in [1]. It contains a textual description of the GMP in natural language, and a more precise description under the form of pseudo-code. We now present the issues that were raised during the review.

3.1. General Properties

The definition of the eight properties of the GMP was carefully reviewed.

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 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 Eventual Integration* cannot be practically verified.

It is worth noting that the formal specification of such properties would be specifically challenging. 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.

3.2. 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 [1], respectively in Sections 3 and 5. They are reproduced below. 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.

$r \le R - 2v(t + t')$	$d_s = R - 2V_{max}(t_u + 7t_d)$
R: range of communication	v, V_{max}: maximum speed
t: time of application level task	t_u: location report period
t': time for a group operation	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 comes from the time needed for a split ($2t_d$) taken into account the uncertainty of nodes (t_u+t_d) and that a merge could be in progress which cannot be

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*.

aborted ($4t_d$), is only *t'* from the left formula, and the *t* part is missing.

The definition and calculation of t_d are also ambiguous. In Section 1 of the paper, parameter t_d is referred 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. For example, if a merge is in progress after the group change notification is received, the processing of messages is blocked.

As can be seen, the calculation of the safe distance is a very complex problem and probably needs more attention.

3.3. 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.

3.4. 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 the safe distance calculation. We are back to the problem of the identification of worst-case scenarios combining mobility patterns with distributed execution schemes.

4. Review of the implementation

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 synchronization problems).

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. Thus, a UML model was created to ease the understanding of the architecture and the behavior of the implementation.

To obtain the static structure, the Java code was reverse engineered into UML class diagrams. This helped us to identify which entities store which part of the state data. To capture the dynamic behavior, sequence diagrams were created for the important scenarios. It helped us to analyze how the implementation works actually, what kind of messages are exchanged during the group change operations. The review of the model revealed the following issues.

4.1. The code differs from the specification

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, but in most cases without success. The most significant differences in the two protocols are the following.

In the specification, 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. The implementation uses a decentralized approach. Each node monitors the location of its neighbors. Connectivity changes (as opposed to precise location information) are reported to the leader only if needed.

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.

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).

A more detailed presentation of the differences can be found in [3]. Taking into consideration all of them, we came to the conclusion that although the two GMP versions are based on the same concept and are characterized by the same properties, they shall be treated as two different protocols. A consequence is that, in our test experiments (Section 5), the pseudo-code description cannot be used as a reference for oracle checks. Pass or Fail verdicts will only be established with respect to the high-level properties in Table 1.

4.2. Atomicity in group change operations

As said before, there are no flush messages in the implementation. In Section 4 of [1], 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 synchronization 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).

4.3. 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 (Fig. 1).

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 and violates the safe distance concept.

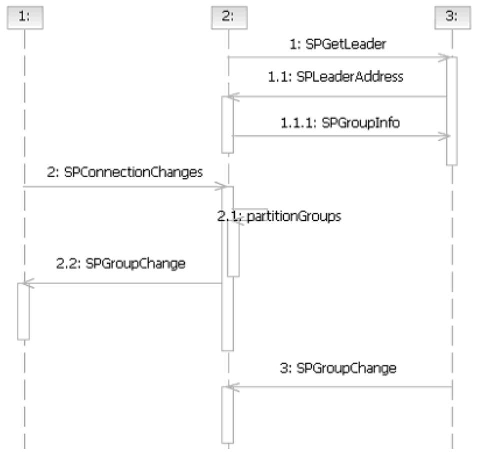

Figure 1. Split interleaving with a merge. Message SPGroupChange contains the new membership view.

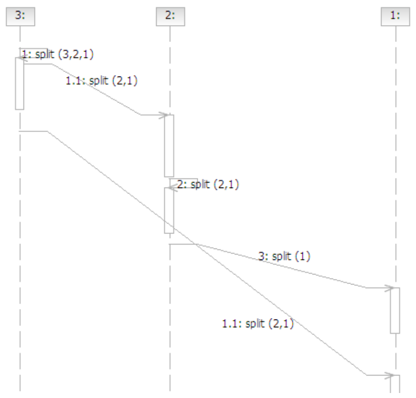

Figure 2. Accepting future messages. Labels *split(new members)* denote SPGroupChange messages.

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

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 far later the message 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.

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.

5. Testing the GMP

Testing is performed against the high-level requirements of the GMP. Since two of the required properties (*Conditional Bounded Integration, Conditional Group Split*) are lacking a precise definition, they are not retained for testing. Two others (*Local Monotonicity, Membership Agreement*) depend on the availability of group identifiers, which is an unimplemented feature. We then slightly modified the source code to introduce it in accordance with the paper specification [1]: each group is identified by a pair (leader id, group change sequence number). We describe below the test platform (Section 5.1) and system parameterization (Section 5.2) used in our experiments. Test results are presented in Section 5.3. They are followed by a discussion in Section 5.4.

5.1. 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 simply use the prototyping facilities offered in the source code distribution.

At the top of the resulting platform (Fig.3), a test *driver* creates a random number of nodes. All nodes physically reside in the same processing unit, and inter-node communication is 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 (xy) 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 compare it 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.

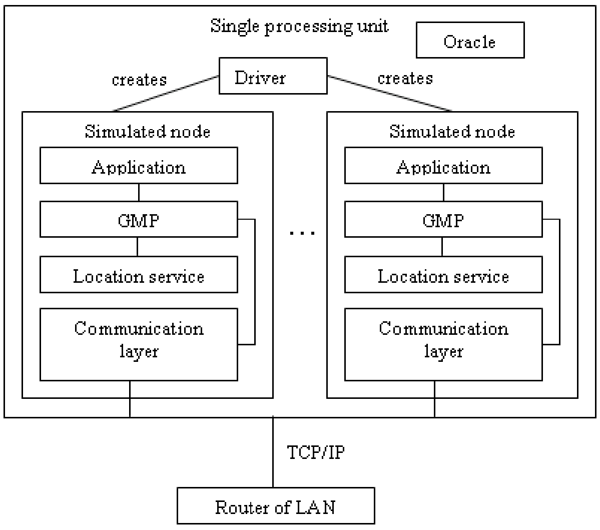

Figure 3. Architecture of the test platform.

5.2. Parameters of system configuration

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 maximal speed of nodes.
- Parameters that are more related to test strategy, like the mobility model used to generate (xy) coordinates for the nodes.

Table 2 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 test parameters, it is worth noting that the GMP behavior (in terms of split and merge operations) is only *indirectly* controlled via the generated node positions.

5.3. Test results

The experiments involved 100 runs, 82 of which violated at least one property. Violations were for:

- Local properties, *Local monotonicity (LM)* and *Membership change justification (MCJ)*.
- Global properties involving several nodes, *Membership Agreement* (MA) and *Same View delivery* (SVD).

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

Table 2. Parameterization of our test experiments

GMP parameters	Value
Transmission range	300 m
Maximal speed	10 m/s
Period for location update	1 s
Network latency	1 s
Test 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

run, by analyzing the first violation and then by searching later scenarios violating new properties. A total amount of 164 scenarios were extracted. As shown in Table 3, the diagnosed problem is always the non-atomicity of merge operations. A merge 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.

An example of concurrent split and merge with multiple violations is given in Figure 4. It is a variant of the scenario found by reviewing the implementation (Fig. 1). In the original scenario, a group leader performs a split operation while being explicitly engaged in a merge operation. In the variant of Figure 4, the 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.

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 4 only represents 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 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.

Table 3. 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 # of scenarios	67	97

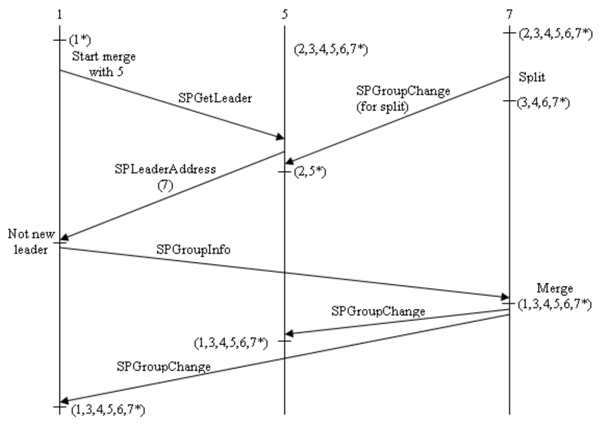

Figure 4. Split and merge interleaving: MCJ+SVD violation.

- 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. 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.

5.4. Discussion

The test experiments could not produce concrete instances of the second scenario found by reviewing the implementation (Fig. 2). 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 2, 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. In the platform, there is no global view of the location of nodes, and no possibility to tune the communication layer to the current topology. Scenarios impacting the safe distance concept were identified by post-mortem analysis (e.g., the scenario in Figure 4 yielding the reintegration of Node 5), but a richer test platform would be required for further investigation of the GMP dynamics.

6. Related work

Related work includes work on modelling mobile systems, on transferring model-based testing technology to this new application field, and on developing test platforms for experimental studies.

6.1. Modelling mobile computing systems

The reverse engineering exercise focused on the code executed by each node. It would have been interesting to produce models giving a global view of sets of nodes and their mobile computing environments.

In the recent years several methods were proposed to model the different aspects of mobile computing, however, no standard has been published yet. Mobile agents gained much attention [4], in [5] a UML profile was defined for global systems, and [6] developed a new formalism, MobiCharts, for mobile computing environments. However, all of the above papers focus more on scenarios, where mobile entities could move to predefined locations, and not on ad hoc networks.

6.2. Model-based Testing

The contributions on model-based testing mainly came from the protocol testing community. In this community, a widely used modelling language is SDL (Specification and Design Language), and there are methods to generate test cases from an SDL specification and a set of test purposes.

In [7], the authors study MIPv6, a protocol enabling nodes to remain reachable while moving around in the IPv6 Internet. In [8], 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 [7] and [8] had to tackle the problem of not having mobility-related concepts into SDL. They had to add specific components (one for each node, plus a centralized controller) to capture the notion of communication with neighbors. They also had to choose a baseline configuration for test generation, e.g., in [8], five nodes connected according to a certain topology, with one link becoming broken after 15 seconds. Moreover, the test approaches focus on conformance with respect to the operational behaviour of the specification (in terms of message emission and reception). They do not consider declarative properties like the local and global ones required from the GMP.

6.3. Test platforms

Test platforms must offer facilities to experiment in mobile settings. The usual approach to simulate wireless communication (in both infrastructure and ad hoc modes) is to integrate a network simulator into the test platform. Such simulators have been originally developed to support networking research, but they are now also used to experiment with the applicative level as well. For example, the ns-2 simulator[1] is used in [9] to evaluate a health-monitoring application, and in [10] to evaluate a car-to-car messaging system. In both cases, the network simulator is only part of the complete platform (see Fig. 5), which also includes a context controller. The context controller is used to simulate contextual information, 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.

As can be seen, technological solutions exist to build simulated environments for mobile applications and middleware. The schematic architecture in Figure 5 yields much more complex platforms than the one we used for first experiments. Such a complexity is probably needed in many cases, so as to be able to test relevant behavior patterns (see our previous examples of untested GMP patterns).

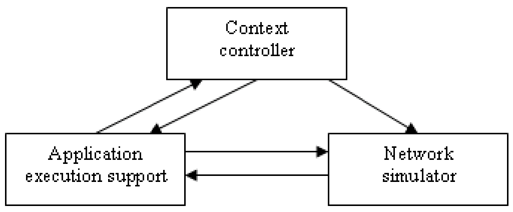

Fig. 5. High-level view of the platforms used by [9, 10]

[1] http://www.isi.edu/nsnam/ns/

7. Conclusion

Our study shows that rigorous software engineering techniques would have been useful to support the design of a complex protocol such as the GMP. Many problems were identified without any preliminary execution of the code. Analyzing the high-level requirements from the perspective of verification yielded the conclusion that some key properties (conditional bounded 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 crude 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. The formalisms do not offer concepts to express the spatiotemporal relationships of nodes as first-class entities. Such concepts would be useful to describe the joint behavior of the nodes as well as the dynamic evolution of the system structure.
- The ability to perform model-based testing crucially depends on the availability of solutions to the previous modeling issue. In the GMP example, we experimented with a crude random strategy for lack of better guidance. The oracle checks were implemented based on our interpretation of the informal high-level requirements. Clearly, the test selection and test oracle problems deserve a substantial amount of work in close connection with work on specification and design methodologies.
- The implementation of testing also raises the technological problem of the test platforms to be settled. 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. Then, an open issue concerns the language to describe concrete scenarios to be implemented on the platform, and the relation with more abstract descriptions at the model level.

We believe that the development of challenging case studies, such as the GMP in this paper, will play a crucial role in guiding investigation to these problems.

Acknowledgements – This work was partially supported by the ReSIST Network of Excellence (IST 026764) and the Hidenets project (IST 26979) funded by the European Union under the Information Society Sixth Framework Programme.

References

[1] Q. Huang, C. Julien, and G. Roman, "Relying on Safe Distance to Achieve Strong Partitionable Group Membership in Ad Hoc Networks", *IEEE Trans. on Mobile Computing,* 3(2), April 2004, pp. 192-204.

[2] G.P. Picco, A.L. Murphy, and G.C. Roman, "LIME: A Middleware for Logical and Physical Mobility", *Proc. 21st Int. Conf. Distributed Computing Systems*, IEEE CS Press, USA, July 2001, pp. 524–536.

[3] Z. Micskei *et al.*, "Analysis of a group membership protocol for Ad-hoc networks", LAAS Report no. 06797, France, November 2006.

[4] E. Belloni and C. Marcos, "MAM-UML: An UML Profile for the Modeling of Mobile-Agent Applications", *Proc. of the XXIV Int. Conf. of the Chilean Computer Science Society (SCCC'04)*, IEEE CS Press, Chile, November 2004, pp. 3-13.

[5] H. Baumeister *et al.*, "UML for Global Computing", *Proc. of Global Computing 2003,* LNCS 2874, Springer-Verlag, Italy, February 2003, pp. 1-24.

[6] S. Acharya and R.K. Shyamasundar, "MOBICHARTS: A Notation to Specify Mobile Computing Applications", *Proc. of 36th Hawaii Int. Conf. on System Sciences (HICSS-36)*, IEEE CS Press, Hawaii, January 2003, pp 298-308.

[7] F. Ngani 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, June 2005, pp. 619-626.

[8] A. Cavalli *et al.*, "A validation Model for the DSR protocol", *Proc 24th Int. Conf. on Distributed Computing Systems Workshops (ICDCSW'04)*, IEEE CS Press, Japan, March 2004, pp.768-773.

[9] R. Morla and N. Davies, "Evaluating a Location-Based Application: A Hybrid Test and Simulation Environment", *IEEE Pervasive computing, Vol.3, No.2*, July-September 2004, pp.48-56.

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