Testing Mobile Computing Applications: Toward a Scenario Language and Tools

Minh Duc Nguyen Hélène Waeselynck Nicolas Rivière
LAAS-CNRS ; Université de Toulouse ; 7, avenue Colonel Roche, F-31077 Toulouse, France
E-mail: {mdnguyen,waeselyn,nriviere}@laas.fr

ABSTRACT
Advances in wireless networking have yielded the development of mobile computing applications. Their unique characteristics (dynamicity of the system structure, communication with unknown partners in local vicinity, context dependency) provide new challenges for verification. This paper elaborates on the testing technology. As a first step, a review of the state-of-the-art is performed together with a case study (a group membership protocol in mobile ad hoc settings), which allowed us to gain insights into testing problems. Work is then directed toward: (1) the definition of a scenario language with extensions to better account for mobile settings (spatial relationships, broadcast communication with neighbors), and (2) an automated support for the off-line analysis of execution traces to identify occurrences of described scenarios.

Categories and Subject Descriptors
D.2.5 [**Testing and Debugging**]: Testing tools

General Terms
Algorithms, Languages.

Keywords
Testing, mobile computing systems, scenario language, graph matching.

1. INTRODUCTION
Advances in wireless networking technology have yielded the emergence of the mobile computing paradigm. Generally speaking, mobile computing calls for the use of mobile devices (e.g., handsets, personal digital assistants, laptops) that move within some physical areas, while being connected to networks by means of wireless links (e.g., Blue-tooth, IEEE 802.11). 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.

From a theoretical and technological point of view, mobile applications are more complex than their non-mobile-counterparts, since they involve not only aspects of distributed applications (e.g., concurrency, communication, latency), but also unique characteristics of mobile systems, such as frequent connections and disconnections of mobile nodes, ad hoc networking, and dependency on context. These novelties in mobile systems let us argue for the necessity of new validation methodologies. The objective of our work is to develop solutions for testing mobile-based applications and middleware services.

As a first step, a review of relevant literature has been performed together with a case study, a group membership protocol (GMP) in ad-hoc networks [12], that allowed us to gain concrete insights into testing 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) the development of an automated support to analyze execution traces, in order to identify occurrences of described scenarios. In the latter case, an occurrence may correspond to the violation of a functional requirement or to the coverage of a test purpose.

This paper is organized as follows. Section 2 recalls the state-of-the-art in testing (traditional) distributed systems. To illustrate the specificities of mobile computing systems, Section 3 introduces the GMP case study and discusses its outcomes in relation with the state-of-the-art. Section 4 argues that graphical scenario languages, widely used to support the design of testing in distributed systems, need extensions to conveniently represent interaction scenarios in mobile settings. Section 5 presents initial work toward the automated processing of the extended scenario descriptions. Section 6 concludes the paper.

2. 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 of Observation (PCOs) to which test components are attached. Synchronization procedures are required to coordinate those components. Recently, TTCN-3 (Testing and Test Control Notation Version 3) [27] has emerged as a powerful language dedicated to the specification and implementation of test cases. Compared to its previous versions, TTCN-3 has improved

WODA – Workshop on Dynamic Analysis, July 21, 2008

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) [13] and their derivatives [5, 24]. They are widely used to represent scenarios of interaction in distributed systems. Indeed, the representation of scenarios may serve different purposes: capture of requirements [16], specification of test purposes (that is, of interaction patterns to be covered by test cases) [8], design of test cases [22], or analysis of execution traces [2]. Their popularity is due to their user-friendly syntax, which facilitates communication while opening the door for formal treatments (this however requires that the used notation is given a precise semantics).

Model-based test generation has been extensively investigated in the framework 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 accepting SDL models, both commercial [15, 14] and academic [7, 3, 25]. 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 fulfill 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., [11]), 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 (see e.g., [17]) may be the most applicable approach in the case of complex systems. It consists in running the system with a real or synthetic workload, in monitoring input and output events, and in analyzing the recorded trace to check for violation of properties.

We will now discuss how mobile settings particularize the testing problems. It is worth noting that there exists two notions of *mobility*. The first one is the logical mobility of code, yielding *mobile computation*. The second one, as mentioned in the introduction, refers to the physical mobility of devices, which defines *mobile computing*. This paper is concerned with mobile computing. A case study is introduced for illustrative purposes.

3. A CASE STUDY IN THE AD HOC DOMAIN

A group membership protocol (GMP) is a fundamental service lying at the heart of fault-tolerant systems. It aims to maintain a consistent view of who is in the group, in spite of the faults that may affect some nodes. The problem has been extensively investigated for traditional distributed systems. However, it needs to be revisited in the case of mobile systems. The authors of [12] propose a solution in the ad hoc domain.

3.1 Overview of the GMP

The goal of this GMP is to offer a membership service that accommodates the disconnections induced by mobility (when mobile nodes get out of range). Each group is managed by a leader that may decide to merge with other groups, or to split its group. Decision is based on the notion of *Safe Distance*. If two nodes are "close enough", this will prevent motion-induced disconnection for some time. A group is then safe if any two members are connected via a path along which all consecutive hosts are at a safe distance (multi-hop communication is allowed). When two groups can merge into a single safe group, they have to do so. When a group is no longer safe, it has to split. The safe distance criterion ensures that there is time to carry out the configuration change before a disconnection occurs. The calculation of the safe distance is based on various parameters such as the transmission range of mobile nodes, their maximal speed, and the time needed for a group level operation. Departing or arriving nodes are detected by having nodes broadcast their physical location ("hello" message).

The authors of the protocol put strong assumptions on the environment (e.g., reliable communication, bounded delays). The membership service is then characterized with the properties presented in Table 1 (see [12] for more information) that should hold whenever the assumptions hold.

An open-source implementation of the GMP is given in LIME, a middleware for mobile systems [19]. The implementation is not just a small example program: each node consists of about 4000 lines of Java code, contains 22 classes and involves 6 concurrent threads.

Table 1. General properties of the protocol

Property	INFORMAL DEFINITION
Self inclusion (SI)	A node is always a part of its membership view.
Local monotonicity (LM)	Group identifiers installed on each node are in increasing order.
Membership agreement (MA)	If two nodes have the same group id, then they have the same membership view.
Initial membership view (IMV)	A node always installs itself as the only member in its view when it starts.
Membership change justification (MCJ)	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 (SVD)	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 (CEI)	If two groups satisfy the merging criteria and do so for long enough, they will merge into one group.
Conditional group split (CGS)	A group splits only if it is necessary.

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

Our analysis of the GMP [26] involved several steps, including a review of the paper specification, and the reverse engineering of the source code to produce UML models. We report here mainly on the test experiments that were also performed.

3.2 Testing the GMP

The test experiments performed in [26] can be seen as a form of passive testing: the GMP is run using a synthetic workload (random movement of nodes at maximal speed), execution traces are collected, and it is automatically checked whether the implementation satisfies its high-level requirements listed in Table 1. The test platform ensures that the GMP environment fulfills the assumptions made by the authors.

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).

A sample of 164 scenarios was extracted for further analysis. As commented out in [26], 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.

3.3 Insights from the study and related work

The test results show that the GMP implementation is flawed. However, our aim in [26] 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. From this perspective, we believe that the GMP is a challenging example. It is representative of a fundamental problem (membership agreement) in distributed computing. It involves ad-hoc networking, a specificity that makes mobile systems depart from the traditional view of distributed systems. The GMP also exhibits a high dependency on the location and movement of nodes.

For such systems, an issue for test platforms is how to accommodate the wireless technology and the mobility of nodes during testing. In [18], a telephony application was tested by having human operators carry handsets in an urban area. In [6], testing a car-to-car application involved three prototype vehicles driven on a road. But it is obvious that such kind of testing can only be limited. In practice, a major part of the testing activities has to be performed using emulation/simulation facilities. Our test platform was based on simple facilities offered in the GMP source code distribution. This was sufficient for quick feedback on the implementation behavior, but suffered from limitations. A conclusion in [26] was the need for more advanced facilities, including:

- A context simulator, to manage the relative position of nodes according to some mobility model and to produce context data (e.g., location-based data) needed by the application.
- A network simulator to control the delivery of messages based on the context (e.g., radio communication in a local vicinity).

Examples of platforms integrating both categories of tools can be found in [20, 23].

Another conclusion of our study was the need for adequate formalisms to support V&V activities for mobile systems. Previous work from the protocol testing community has shown that it is possible to cope with classical formalisms, to some extent (SDL models of mobility protocols were used in [4, 21]). However, this is done at the expense of modeling tricks. The authors added specific components to capture the notion of communications with neighbors. They also had to choose a baseline configuration for test synthesis.

It would be more convenient to work from specification and design models equipped with primitive concepts for representing mobile settings. In the GMP example, the reverse engineering exercise showed that standard UML might be sufficient to represent one node in isolation, but that system-level behavior and structure are not easily captured [26]. Some UML extensions have been proposed in [1, 9] to represent location and mobility. But the offered concepts mostly consist in entering and exiting boxes, which may be convenient to represent the movement of a node from one infrastructure access point to the other, but does not cover applications with ad hoc networking.

From a testing perspective, relevant formalisms may range from scenario languages to languages for modeling complete behavior. In this paper, we investigate scenario languages. We propose a set of extensions to account for mobile settings, and present initial work toward the automated processing of scenario descriptions.

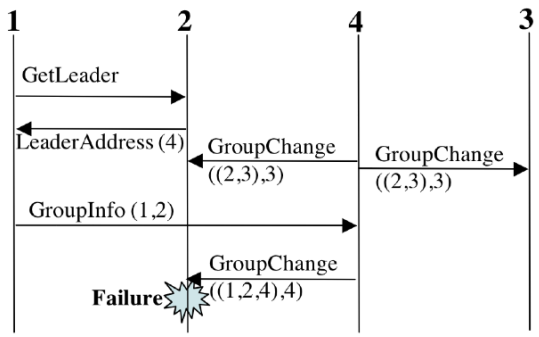
4. SCENARIO LANGUAGES IN MOBILE SETTINGS

Scenarios 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. For illustrative purposes, Figure 1 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 (see Table 1) violation occurs on Node 2.

In the GMP case study, such graphical representations proved crucially useful to understand the fail behavior during testing. We reformulated part of the monitored traces into MSC-like diagrams to get a clearer picture of what was happening.

Graphical representations of scenarios were also useful at the previous phases of the study, when analyzing the specification and the design. Still, we found that classical scenario languages could not capture some specificities of mobile computing applications.

Let us come back to the example in Figure 1. First, it puts emphasis on the partial order of messages, while the spatial topology of nodes is equally important to characterize the scenario. Indeed, the split and merge behavior is governed by perceived changes in the topology. 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 for lack of convenient language constructs. Note that the sender of an “hello” message does not *a priori* know the number and identity of potential receivers. Whoever is at communication range may process the message.



Messages in GMP

- GetLeader: Asks for the current leader of a new node detected at a safe distance.

- LeaderAddress (*leader*): On receiving a GetLeader message, a node replies by sending its leader address.

- GroupInfo (*connectivity list*): During a merge operation, a node sends its connectivity information to the future new leader.

- GroupChange (*members*, *leader*): The old leader sends out GroupChange messages containing new members and new leaders after performing a merge or a split.

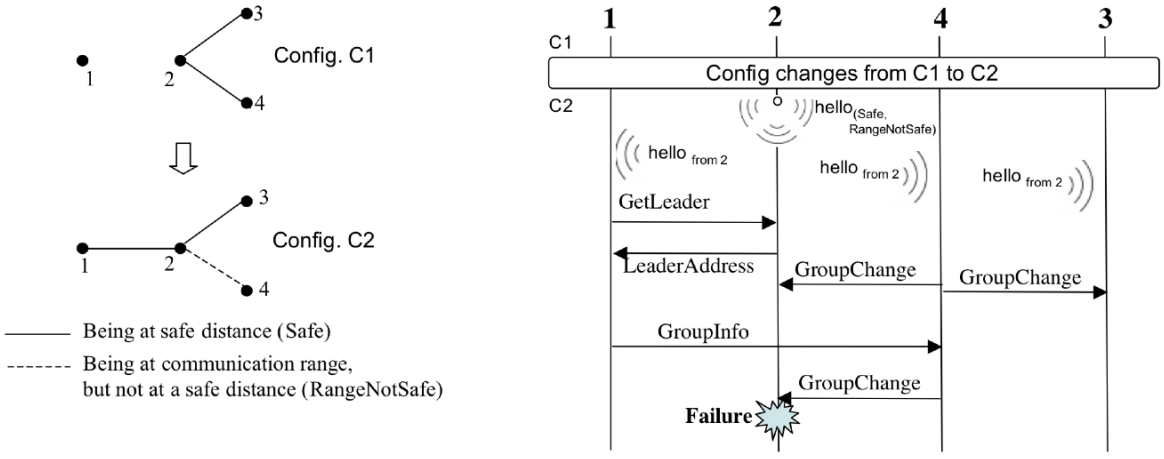
Figure 1. MSC-like view of a fail scenario

Let us know consider the extended representation shown in Figure 2. It contains both (a) a graph-like view of the successive spatial configurations, and (b) 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 2 after a change in spatial configuration. Specifically, Node 2 is no longer at a safe distance from Node 4, while getting close to Node 1. Both the GetLeader message from Node 1 (initiating a merge) and the GroupChange message from Node 4 (performing the split) are causally related to the "hello" message: when receiving the location data from 2, the other nodes get aware of a change in the topology. Without entering the details of the scenario, let us insist on 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 transitive safe path between 4 and 3 (it is precisely the inconsistent treatment of Node 3 in the group change operations received by 2 that will induce the MCJ violation);
- When asked about its leader address, Node 2 replies before being informed about the split.

Abstracting from the concrete syntax shown in Figure 2, the language extensions to conveniently represent scenarios for mobile computing systems are 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 decision to split or merge groups), and being at communication range (which determines the single-hop connectivity). Labeled graphs are adequate to capture such 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 synchronization (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 2b, this is concretely 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.



(a) Spatial configuration: Graph-like view **(b) Events: MSC-like view**

Figure 2. Scenario mixing spatial and event views

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 (capture of application requirements, specification of test purposes, of test cases, reverse engineering from execution traces). As an example, we are currently working on the extension of UML 2.0 sequence diagrams, in close collaboration with the Budapest University of Technology and Economics. Much attention is paid to semantics issues, so that the formal treatment of scenario descriptions becomes possible.

5. PROCESSING OF SCENARIO DESCRIPTIONS

As already mentioned, scenario descriptions may be a useful support to various V&V activities, ranging from the capture of application requirements to the specification and design of test cases. In some cases, the scenarios may be used in conjunction with a complete model of behavior. For example, it is formally verified whether the model may violate a requirement scenario. Or test cases are formally synthesized from the model and a set of test purpose scenarios. But the usefulness of scenarios does not require commitment to complete formal specification. Indeed, some automated treatments become possible by simply using scenario descriptions (provided the semantics is well-defined). They include checking whether a test execution trace satisfies a requirement scenario, or whether a test execution trace covers a test purpose.

These treatments involve classical treatments in scenario-based frameworks, such as the comparison of the orders of events in two scenario representations. In mobile settings, the proposed extensions now consider the spatial relationships of mobile devices as first class concepts. We then argue that graph algorithms, and more specifically algorithms for graph matching problems, are also needed 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 below.

5.1 Relevance of Graph Matching Problems

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

Generally speaking, the location data may have to obey a complex mobility model (for example, cars moving in a geographical area, with a given network of roads). Also, the abstractions used in the labeled graph may be richer than just the distance of nodes. As a result, the manual production of context data may be cumbersome, if not unrealistic.

Although we may not be able to manually tune data, it may not be too difficult to trigger the desired configuration at least once by random simulation. In test platforms, context simulators are used to manage the movement of nodes and produce context data accordingly (see Section 3.3). Our proposal is then to have a preliminary production phase of context data, based on runs of the context simulator taken in isolation:

- A run may involve numerous 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 graphs representing the evolution of the system topology.
- It is then searched whether sub-graphs can match the desired evolution pattern (defined in the scenario).
- The list of matches provides alternative baseline configurations for the implementation of the scenario.

For example, from a simulation run, we may identify four nodes exhibiting the spatial configurations shown in Figure 2a. 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.

Graph matching algorithms are also relevant to comparing a test execution trace to either a test purpose (in order to detect coverage), or an application requirement (in order 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.

5.2 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.

A graph homomorphism is a mapping between two graphs that respects their structure. It can be mathematically defined as follows.

Definition. 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:

- It is injective,
- $\lambda_1(v_1) = \lambda_2(f(v_1))$ for all $v_1 \in V_1$,
- 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 $\mu_1(e_1) = \mu_2(e_2)$.

The definition captures the idea of G_1 being matched by a subgraph of G_2. Usually, G_1 is called the *model graph*, and G_2 the *host graph*. In our work, the model graph is expected to come

from a scenario description, while the host graph is extracted from an execution or simulation trace.

In practice, the basic definitions of graph structure and graph homomorphism need to be slightly extended to fulfill 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, assume that an application involves both mobile and infrastructure mode. A node could be characterized by a 2-tuple <id, type>, where id would be a value uniquely identifying the physical node, and type would be an element of {Mobile, Infrastructure} that differentiates the mobile and infrastructure nodes. Second, 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 <"10", Mobile> with substitution n1:= "10". As can be seen in this example, introducing variables means that the graph homomorphism building needs to exhibit a valuation that consistently unifies the labels.

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 at LAAS-CNRS [10] in the framework of research on dynamically reconfigurable architectures. The tool searches for the set of all homomorphisms (f,Val) from a model graph G_m to a host graph G_h, where f is a mapping and Val is a valuation. In the definition of graph structures, the tool offers the following features:

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

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 tool also offers other facilities that are convenient for us. For example, the function VALUATE_VERTEX(G,Val) takes as inputs a graph and a valuation, and rewrites all vertices according to the valuation (see [10] for the full functionality of the tool).

5.3 Algorithm for analyzing a simulation run

We retrieved this tool and have used it as a basic element to develop an algorithm for the analysis of execution traces and described scenarios, as discussed in Section 5.1.

The algorithm 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 an homomorphism *(f, Val)* from G_{m1} to G_{hi}
- there is an 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. This is so because 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 consider all possible matches for the n simulation steps, which adds to the complexity 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.

6. CONCLUSION AND PERSPECTIVE

This paper elaborates on testing technology. It discusses some issues in testing mobile computing applications, from both technological and conceptual viewpoints. The issues are exemplified by the GMP case study. We then present orientation of our work towards the definition of a scenario language that aims to better account for mobile settings. This language is based on graphical scenario languages with the following extensions:

- 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 a local vicinity is introduced by means of special symbols.

The language has been informally presented, and we are currently working on a precise definition of both the syntax and the semantics.

Since scenario descriptions now involve graph constructs, their formal treatment has to include graph matching algorithms. We have started to develop an algorithm, based on an existing graph tool, to search for the matches of scenarios in a simulation run. Future work will investigate how to accommodate richer descriptions of the node configurations in the scenarios, such as the introduction of min, max duration constraints for the configurations. For example, a scenario may require that a given configuration lasts at least k steps.

Our ultimate goal is the development of a scenario-based testing framework, which we believe would be a pragmatic contribution to the difficult issue of testing mobile computing applications.

7. 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 Program.

8. REFERENCES

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

[2] 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.

[3] A. Cavalli *et al.*, "Hit-or-jump: An algorithm for embedded testing with applications to IN services", *Proc. of the IFIP Int. Conf. FORTE/PSTV '99*, Elevier, Oct. 1999, pp. 41-56.

[4] 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.

[5] W. Damm and D. Harel, "LSCs: Breathing life into message sequence charts", *Proc. of the 3rd Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, Kluwer Academic Publishers, Italy, Feb. 1999, pp. 293-312.

[6] D. de Bruin *et al.*, "Design and test of a cooperative adaptive cruise control system", *Intelligent Vehicles symposium*, IEEE CS Press, Italy, June 2004, pp. 392-396.

[7] J.-C. Fernandez *et al.*, "Using on-the-fly Verification Techniques for the Generation of Test Suites", *Proc. Conf. on Computer-Aided Verification (CAV'96)*, LNCS 1102, Springer Verlag, Aug. 1996, pp. 348-359.

[8] J. Grabowski, D. Hogrefe and R. Nahm, "Test Case Generation with Test Purpose Specification by MSCs", *Proc. of the 6th SDL Forum*, North Holland, Oct. 1993, pp. 253-166.

[9] V. Grassi *et al.*, "A UML Profile to Model Mobile Sytem", LNCS 3273, Springer-Verlag, 2004, pp. 128-142.

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

[11] A. Hartman and K. Nagin, "The AGEDIS tools for model based testing", *Proc. of the ACM/SIGSOFT Int. Symp. on Software Testing and Analysis (ISSTA 2004)*, ACM Sigsoft SE notes 29(4), July 2004, pp. 129-132.

[12] 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, pp. 192-205.

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

[14] A. Kerbrat *et al.*, "Automated test generation from SDL specifications", SDL Forum, 1999, pp. 135-152.

[15] B. Koch *et al.*, "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.

[16] H. Kugler *et al.*, "Testing Scenario-Based Models", *Proc. Fundamental Approaches to Software Engineering (FASE'07)*, LNCS 4422, Springer-Verlag, 2007, pp. 306-320.

[17] B.T. Ladani, B. Alcalde and A. Cavalli, "Passive testing – a constrained invariant checking approach", *Proc. 17th IFIP Int. Conf. on Testing of Communicating Systems*, LNCS 3502, Springer-Verlag, 2005, pp. 9-22.

[18] 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*, IEEE CS Press, Korea, Dec. 2004, pp. 430-437.

[19] LIME, URL: http://lime.sourceforge.net

[20] R. Morla and N. Davies, "Evaluating a Location-Based Application: A Hybrid Test and Simulation Environment", IEEE Pervasive computing 3(2), Jul.-Sep. 2004, pp. 48-56.

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

[22] S. Pickin and J-M. Jézéquel, " Using UML sequence diagrams as the basis for a formal test description language", *Proc. of 4th Int. Conf. on Integrated Formal Methods*, LNCS 2999, Springer-Verlag, 2004, pp. 481-500.

[23] C. Schroth *et al.*, "Simulating the traffic effects of vehicle-to-vehicle messaging systems", *5th Int. Conf. on ITS Telecommunications (ITST 2005)*, France, June 2005, pp. 155-179.

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

[25] G.J. Tretmans and H. Brinksma, "Côte de Resyste -- Automated Model Based Testing", *Proc. of the 3rd Progress Works. on Embedded Systems*, Netherlands, Oct. 2002, pp. 246-255.

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

[27] C. Willcock *et al.*, An Introduction to TTCN-3, Wiley, 2005.