TERMOS: a Formal Language for Scenarios in Mobile Computing Systems

Hélène Waeselynck[1,2], Zoltán Micskei[3], Nicolas Rivière[1,2], Áron Hamvas[3], Irina Nitu[1,2]

[1]CNRS; LAAS
7 av. Colonel Roche
F-31077 Toulouse, France

[2]Université de Toulouse
UPS, INSA, INP, ISAE ; LAAS
F-31077 Toulouse, France

[3]Budapest University of Technology and Economics
Muegyetem rkp. 3
1111 Budapest, Hungary

{waeselyn, nriviere}@laas.fr, micskeiz@mit.bme.hu

Abstract. This paper presents TERMOS, a UML-based formal language for specifying scenarios in mobile computing systems. TERMOS scenarios are used for the verification of test traces. They capture key properties to be checked on the traces, considering both the spatial configuration of nodes and their communication. We give an overview of the TERMOS design and semantics. As part of the semantics, we present the principle of an algorithm that computes the orders of events from a scenario. Two proof-of-concept prototypes have been developed to study the realization of the algorithm.

Keywords: Mobile computing systems, UML Sequence Diagrams, formal semantics, testing.

1 Introduction

Graphical scenario languages (e.g., Message Sequence Charts [1], UML Sequence Diagrams [2]) allow the visual representation of interactions in distributed systems. Typical use cases, forbidden behaviors, test cases and many more aspects can be depicted. The popularity of graphical scenarios is due to their user-friendly syntax, which facilitates communication while opening the door for formal treatments (this however requires that the used notation has a precise semantics). We investigate one of such formal treatments, namely the automated analysis of test traces. In our work, a scenario captures a key property to be checked on the traces. Scenario-based verification is not a novel approach, the originality here is that we apply it to a specific class of distributed systems: mobile computing systems.

Mobile computing systems involve 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). Such systems differ from "traditional" distributed systems in many aspects: frequent connections and disconnections of mobile nodes, communication with unknown partners in a local

vicinity, context awareness. These novelties require us to revisit the notion of scenario for mobile settings.

Preliminary work led us to conclude that some extensions are necessary [3]. One of these extensions is to consider the spatial relations between nodes as a first class concept. We proposed to use labeled graphs to depict the spatial configurations of a scenario. We then noticed that, due to this extension, the checking of test traces against scenarios has to combine event order analysis and graph matching.

Since the preliminary workshop paper, the graph matching part has been addressed in [4]. This paper now aims at giving an overview of the whole approach, and at presenting the remaining event order analysis. This analysis considers both communication and configuration change events.

The contributions are the following:

- We show how the general extensions proposed in [3] can be introduced into one of the existing scenario languages, UML Sequence Diagrams (Section 3).
- We then define a specialization of Sequence Diagrams, called TERMOS (Test Requirement language for Mobile Setting). TERMOS is given a formal semantics suitable for the checking of traces, at the price of some syntactic constraints and some interpretation choices of UML constructs (Section 4).
- We explain the principle of the algorithm that computes orders of events from a TERMOS scenario according to the chosen semantics (Section 5.1, see also our technical report [5] for a detailed presentation of the algorithm).
- We present two prototypes we developed as a proof-of-concept of the algorithm (Section 5.2).

2 Related Work

Previous work has investigated how to incorporate mobility into UML scenarios [6, 7, 8]. However, the focus was more on logical mobility (mobile computation) than physical mobility (mobile computing). It induces a view of mobility that consists of entering and exiting administrative domains, the domains being hierarchically organized. This view is adequate to express the migration of agents, but physical mobility requires further investigation, e.g., to account for dynamic ad-hoc networking. Also, there is not always a formal semantics attached to the proposed notations.

Having a formal semantics is crucial for our objective of analyzing traces. We had a thorough look at existing semantics for UML 2.0 Sequence Diagrams [9]. We also looked at other scenario languages distinguishing potential and mandatory behavior, which makes them well suited for the expression of properties [10, 11]. The most influential work for the TERMOS semantics was work on Live Sequence Charts (LSC) [10], and more specifically Kloses's version of the semantics [12], as well as work adapting LSC concepts into UML Sequence Diagrams [13, 14].

3 Graphical Interaction Scenarios for Mobile Computing Systems

Graphical scenario languages are used to represent interactions in distributed systems. Lifelines are drawn for the individual participants in the interaction, and the partial orders of their communication events are shown. To represent complex orderings, the languages offer operators like choice, iteration, parallelism and sequencing. As mentioned in Section 2, some languages also distinguish potential and mandatory behavior. A point-to-point view of communication is usually adopted, with one sender and one receiver for a given message. The underlying connection topology is not the focus, it is not supposed to change during the shown interaction.

Such characteristics are not sufficient for mobile computing systems, where the movement of nodes inherently yields an unstable topology. Links with other mobile and infrastructure nodes may be established or destroyed depending on the locations. Moreover, nodes may dynamically appear and disappear as devices are switched on and off, run out of power or go to standby. Interaction scenarios should thus explicitly account for the spatial configuration of nodes and how it evolves.

In addition to usual point-to-point communication, a natural communication for mobile computing systems is local broadcast. In this class of communication, a node broadcasts a message in its neighboring environment. Whoever is at transmission range may listen to, and react to, the message. For example, local broadcast is used as a basic step for the discovery layer in mobile-based applications (group discovery for group membership services, route discovery in routing protocols, etc.).

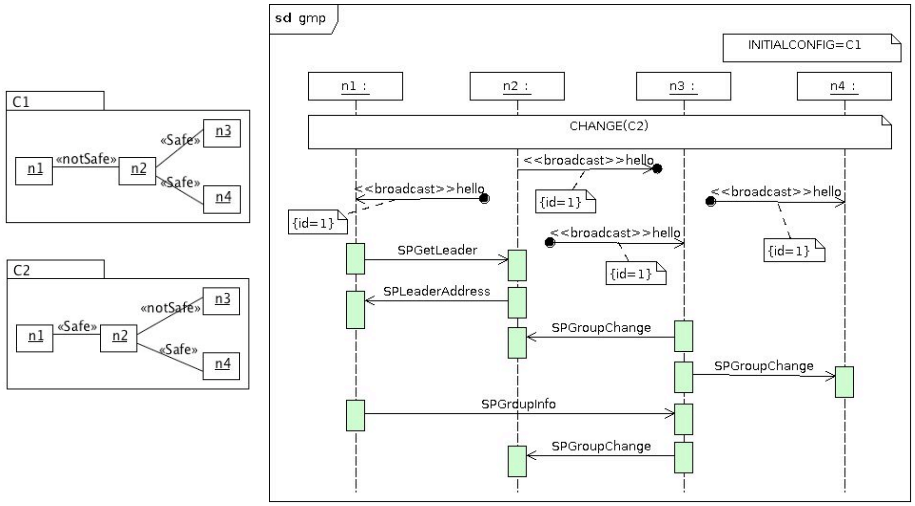
We propose that existing scenario languages be extended to accommodate local broadcast and make the spatial configuration of nodes a first class concept. More precisely, we proposed in [3] the following three extensions:

1. introduction of a spatial view,
2. consideration for spatial configuration change events,
3. representation of broadcast communication events.

The extensions can be introduced into the various existing languages. We now elaborate a solution for UML SD.

3.1 Introduction of a Spatial View

A scenario in a mobile setting contains two connected views: the event view (as classically done), and a spatial view describing the topological configurations of nodes. Fig. 1 illustrates these two views in UML SD. The represented scenario comes from a case study we performed [15], a Group Membership Protocol (GMP) for ad hoc networks. In this GMP [16], groups split and merge according to the location of mobile nodes. The protocol uses the concept of safe distance to determine which nodes should form a group.



(a) Spatial view (b) Event view

Fig. 1. A concurrent split and merge scenario for the GMP

Conceptually, the spatial view consists of a set of labelled graphs, corresponding to the various configurations that occur in the scenario. We depict them using UML object diagrams (Fig. 1a). The shown scenario involves two successive spatial configurations (C1 and C2). Nodes are depicted by instances. They are identified by an id (e.g., *n1*, *n2*, …) and could have contextual attributes (not shown here). Edge labels characterize the connection of nodes. They are represented as stereotypes. In the GMP example, labels indicate whether the nodes are at a safe distance *(«Safe»)*, or whether they are at communication range but not at a safe distance (*«notSafe»)*.

3.2 Consideration for Spatial Configuration Change Events

To connect the spatial and event views, configuration changes are introduced as global events in the event view. The event view in Fig. 1b has an initial configuration (shown in the *INITIALCONFIG* comment box). Configuration changes are then represented by global events of the form *CHANGE(name_of_new_config)*. In the GMP scenario, the two successive configurations correspond to node *n2* getting close to *n1*, while getting away from *n3*.

In other scenarios, configuration changes may involve the dynamic creation and shutdown of nodes. As there is no convenient way to describe such a dynamic structure in sequence diagrams, we adopt the convention that every node mentioned in at least one configuration has a lifeline in the event view. If a node is not active at some point of the scenario, then it is not supposed to participate in any communication interaction. The event view makes it explicit which communication event occurs in which spatial configuration. Checks can then be provided to warn the

scenario specifier whenever communication is not compatible with the spatial view. In Section 5, we will see a tool that includes such checks.

Finally, note that the representation of configuration changes as global events is an abstraction for the fact that the topology is a global system property. A configuration change event does not result from an active synchronization of nodes, it just happens in their physical world. In the GMP example, the change to C2 remains unnoticed until *n2* broadcasts its new location by a *hello* message.

3.3 Representation of Broadcast Communication Events

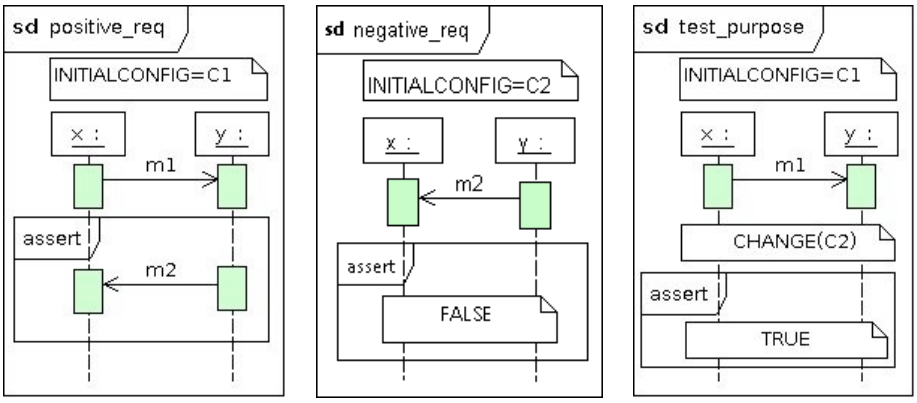
To represent the local broadcast, we use the concepts of lost and found messages and a special stereotype. Lost messages are messages with no explicit receiver. Similarly, found messages do not have an explicit sender. In order to distinguish broadcasts from "usual" lost/found messages, we assign them the «broadcast» stereotype. A broadcast involves one send event followed by one or several receive events. A tagged value is attached to the corresponding lost/found messages, so that each receive event of the diagram can be paired to the send event that caused it.

To conclude, we believe that the three proposed extensions address needs that are recurring when modeling scenarios in mobile settings. They allow us to depict non trivial scenarios, like the concurrent group merge and split of Fig. 1. Test experiments have shown that this interaction yields a GMP failure [15]. The spatial and event views are complementary to describe this fail scenario, which combines a specific evolution of configuration and a specific ordering of messages after the configuration change.

4 Analyzing Test Traces wrt Requirements and Test Purposes

Graphical scenarios may be used at various phases of a development process, ranging from the elicitation of requirements to late testing phases. Our work focuses on the use of scenarios to analyze execution traces of mobile computing systems. We assume that the traces are collected on a test platform offering log facilities. The logged trace contains both communication data and contextual data (e.g., location data) from which the system spatial configurations can be retrieved. We then want to check whether the test trace exhibits some behavior patterns described by scenarios.

We consider three classes of scenarios exemplified by Fig. 2. *Positive requirements* capture key invariant properties of the form: whenever *A* happens in the trace, then *B* always follows. *Negative requirements* describe forbidden behaviors that should never occur in the trace. *Test purposes* describe behaviors to be covered by testing, that is, we would like these behaviors to occur at least once in the trace. The TERMOS language (Test Requirement language for Mobile Setting) is a UML-based notation we developed to capture the three classes of scenarios. It incorporates the extensions previously proposed to account for mobile settings. The notation has a formal semantics, so as to allow the automated analysis of test traces. We introduce here the general principle of this analysis.



(a) Positive requirement (b) Negative requirement (c) Test purpose

Fig. 2. Requirement and test purpose scenarios (event views)

A TERMOS scenario ends with an *assert* fragment. Everything before the *assert* represents a potential behavior, while the content of the *assert* is mandatory. The content of the *assert* characterizes the class of the scenario. A negative requirement contains just a false invariant. Since *FALSE* never holds, this means that the behavior before the *assert* should never happen. A test purpose contains just a true invariant. It holds whenever the *assert* is reached. A positive requirement contains a fragment different from a trivial true or false invariant.

We interpret TERMOS scenarios as generic behavior patterns that may be matched by various subsets of the system during the test run. In Fig.2, the node ids *x* and *y* are *symbolic* node ids. For example, the positive requirement (Fig.2a) is interpreted as:

> "Whenever two nodes exhibit spatial configuration C1, and the node matching *x* sends message *m1* to the node matching y, then the node matching *y* must answer with message *m2*."

At some point of a test run, we may have two simultaneous instances of *C1*, one with system nodes n_1 and n_2 matching *x* and *y*, and one with n_1 and n_3. At some later point, system node n_1 may play the role of *y* in yet another instance of *C1*.

Given a scenario, the analysis of a test trace thus involves two steps:

1. Determine which physical nodes of the trace exhibit the (sequence of) configuration(s) of the scenario, and when they do so.
2. Analyze the order of events in the identified configurations.

Assuming that system configuration graphs can be built from the contextual test data, step 1 amounts to a graph matching problem. We explained in [4] how subgraph isomorphism can be used to search for all instances of the scenario configurations in a trace. We developed a tool, *GraphSeq*, and performed experiments using randomly generated graphs, contextual data from a mobility simulator, and test traces from the GMP case study [4].

In this paper, we focus on the second step of the trace analysis. The principle is to process the event view of a scenario to build a symbolic automaton, having variables

that depend on the spatial configurations. The automaton can then be instantiated according to *GraphSeq* outputs, and used to analyze the order of communication and configuration change events for all found spatial matches.

Before describing the building of the automaton, we discuss some design choices made for the TERMOS language.

5 Design choices for the TERMOS Event View

When developing a new language both its syntax and semantics have to suit the high-level goal of the language. The primary purpose of TERMOS scenarios is to describe various verification artifacts, and to use these scenarios for checking execution traces. Hence the language should be designed to make it possible to determine whether or not a test trace fulfils a target property by (i) limiting non-determinism, and (ii) providing an operational semantics that can be implemented in a toolset.

As TERMOS is based on UML 2 Sequence Diagrams, as a first step the capabilities of Sequence Diagrams were investigated. We surveyed the existing formal semantics proposed for Sequence Diagrams [9]. It turned out that many semantic choices exist even for simple diagrams and elements, e.g., whether a diagram is a complete or a partial interaction or how a trace is represented. If more complex language elements are allowed which can express alternatives or negation, it becomes harder and harder to decide whether a given execution trace conforms to a scenario. Therefore we selected those options for TERMOS from the choices collected in [9] which make checking of traces possible, and added further syntactic restrictions. This section highlights these design decisions (for a complete list see [5]).

- *Interpretation of a basic Interaction:* In TERMOS the execution traces can be categorized as valid, invalid or inconclusive (as opposed to some approaches where only two categories are sufficient). In a requirement or test purpose scenario only the relevant part of the behaviour is depicted (subset of nodes, subset of messages). Hence a TERMOS scenario describes only a partial interaction: a prefix or suffix is allowed in the execution traces and extra messages can interleave.
- *Introducing CombinedFragments:* Alternative fragments, parallel compositions or even negation can be expressed with CombinedFragments. However, in UML there is no synchronization mechanism amongst lifelines when entering or exiting fragments. This could present several challenges when verifying traces (e.g., there is no common point to evaluate guards or the scope of the operator is unclear). For this reason, in TERMOS, we interpret the entering and exiting of a CombinedFragment as a synchronization point for the participating lifelines.
- *Computing partial orders:* The orderings between the elements of the diagrams are encoded in a state-based formalism. A crucial point is how to handle the alternative fragments in the diagrams. In TERMOS to make verification possible there is a global time point when all the participating lifelines evaluate the guards and choose one alternative (this will be represented by a common transition in the formal semantics). Moreover, only a deterministic form of guarded choice is allowed (similar to an if-then-else construct). Finally, variables in guards and state invariants can only refer to message parameters previously sent or received and to

node attributes in the current configuration. This guarantees that unrepresented nodes and messages cannot change the valuation of a predicate.

– *Interpretation of conformance-related operators:* The conformance-related operators (*assert*, *neg*, *consider*, *ignore*) modify the categorization of a trace as valid, invalid or inconclusive. Their usage is heavily restricted in order to make the checking of a trace feasible. Negation can be used only for the whole interaction as a global false predicate. The diagram can have only one *assert* box at the end of the diagram, which should cover all lifelines. *Consider* and *ignore* change the alphabet of messages that are allowed to interleave with the ones depicted in the diagram. In TERMOS, the default interpretation is that any extra message may interleave, hence the *ignore* operator is not needed. We use *consider* to reduce the set of valid traces, i.e. to indicate that some of the extra messages are not allowed. Only one level of nesting is allowed for conformance-related operators (e.g., *assert* into a toplevel *consider*). Furthermore the type of nesting is also restricted (e.g., no double assertion).

We implemented a prototype tool that checks conformance to the syntactic restrictions that TERMOS puts on UML interaction models. The tool is an Eclipse plug-in and can be used to check diagrams imported from a UML editor. In addition to syntactic checks, it performs some semantic verification: for each message, it checks whether the communication between the sender and the receiver can take place in the current spatial configuration (nodes are active and connected by an edge).

6 Semantics of the TERMOS Event View

We now provide a high-level description of the semantics of the TERMOS event view. A detailed, technical description is to be found in [5]. Here, we illustrate the principle by the scenario shown in Fig. 3. The interaction takes place in a hypothetical application where infrastructure nodes with fixed locations periodically broadcast information. Any mobile node that approaches an infrastructure node may ask for additional details, in which case the details must be delivered to it.

For space constraints, the illustrative scenario is kept simple. It contains no alternative or parallel fragments. Obviously, our algorithm for building the automaton accommodates all such constructs from the TERMOS syntax (see [5]).

6.1 Construction of the Automaton

The construction of the symbolic automaton corresponding to the TERMOS event view involves two steps.

Pre-processing: First, the diagram is parsed to extract its atomic elements and the ordering relations between them. For example, the atoms appearing on the infrastructureNode lifeline are: the lifeline head, the configuration change, the sending of information, the receiving of getDetail, the entering of *assert*, the sending of details, the exiting of *assert*, the lifeline end. Atoms are grouped into classes capturing simultaneity. For example, all lifelines enter the *assert* at the same time.

Precedence and conflict relations between classes are then computed. For example, the sending of getDetail precedes its receiving, and both precede the entering of assert. Conflict relations would concern atoms located in different operands of an *alt* CombinedFragment, this is not illustrated by Fig.3.

Unwinding: In the second step, the symbolic automaton is built by gradually unwinding the classes of atoms, until all of them have been processed. A state of the automaton is a global state of the scenario, capturing the progress of all lifelines. The algorithm starts in an initial state with the lifelines heads unwound. Then, it uses the precedence and conflict relations to search for the enabled classes of atoms and to compute the successor states. Each transition is labeled according to the currently unwound class. A label can be an event expression, consuming a trace event that matches it, or a predicate to be evaluated without consuming an event. Both kinds of labels may involve variables, and event consumption may trigger update actions. If the trace analysis reaches a state where no transition can be fired, the automaton exits and returns a verdict that depends on the category of the state.

We illustrate these notions informally on the example.

The automaton of Fig. 4 has three categories of states. Double circle nodes represent trivial accept states: they are used to categorize traces that do not exhibit the potential behavior before the *assert*. Single circle nodes are the reject states (trace is invalid) and triple circle nodes the stringent accept states (trace successfully reaches the end of the *assert*).

In Fig. 4, the states are labelled by an id and the set of variables that are currently valuated. For example, in initial state *0*, the only valuated variables are the ones from the current C1 configuration: we know the identity of the nodes playing the roles of infrastructureNode and mobileNode.

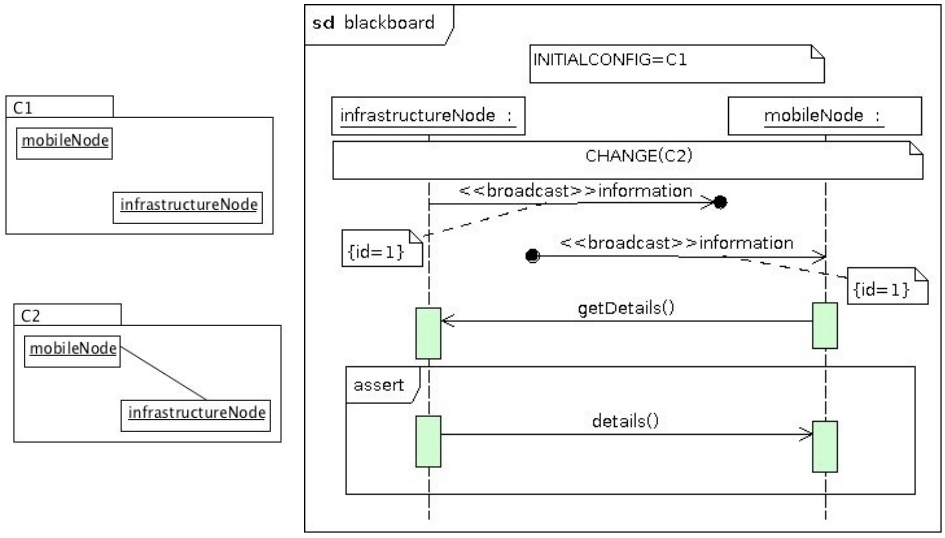


Fig. 3. A TERMOS scenario

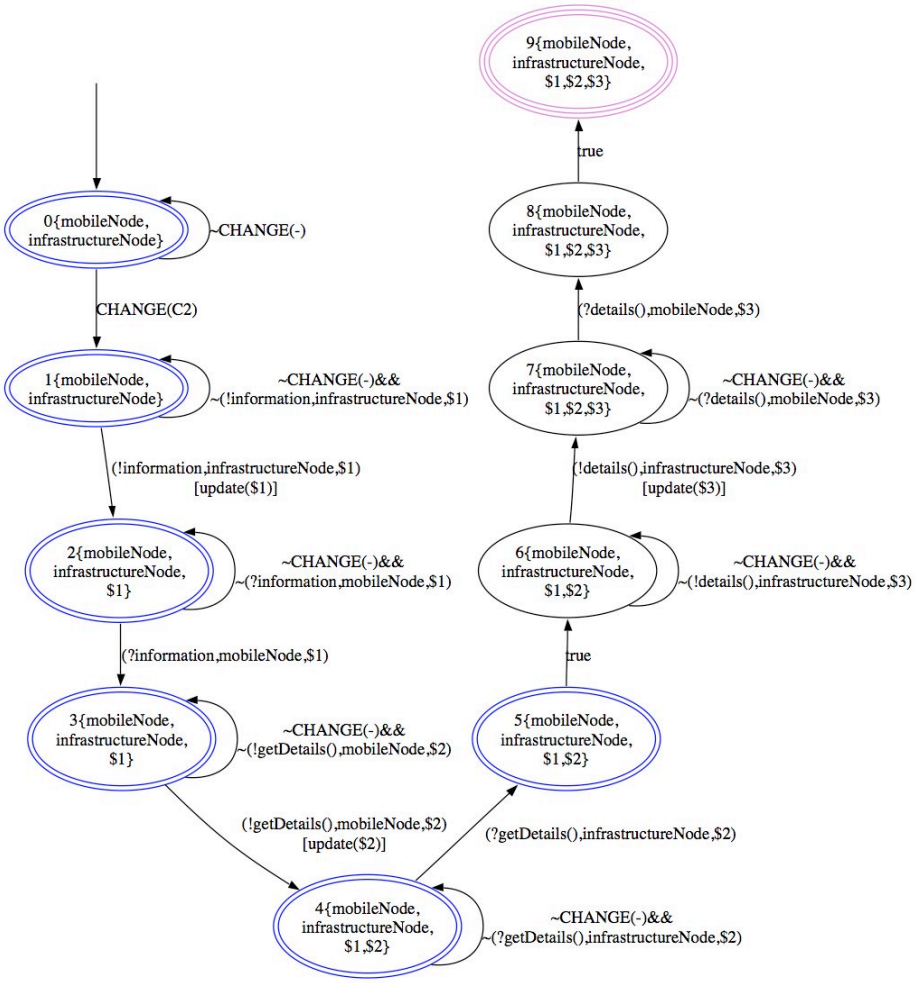


Fig. 4. Symbolic automaton from the scenario in Fig.3

Transitions coming out of states *0* and *1* have labels illustrating event expressions. Transition from *0* to *1* consumes a configuration change to C2. The self-loop on *0* consumes any event that does not match a configuration change (i.e., it consumes any communication event). Transition from *1* to *2* consumes the sending of *information*. The event expression is a triplet, where *!information* denotes the sending of the message, *insfrastructureNode* is the node performing this event, and *$1* is a symbolic message id. Since *$1* is a free variable in state *1*, it can be matched by any id generated by the instrumentation functions of the test platform. Transition to state *2* updates this variable, hence *$1* is no longer free when waiting for the receive event.

Transitions that are not labelled by event expressions do not consume trace events. Examples are *5->6* and *8->9*, corresponding to the entering and exiting of *assert*.

6.2 Proof-of-Concept prototypes

We developed two prototypes, demonstrating the building of the automaton from different standpoints.

The first prototype, developed at LAAS, focused on the study of the algorithm itself. The aim was to convince us that it captures the intended meaning of diagrams. Accordingly, the tool provides a graphical visualization of the generated automaton structure (Fig. 4 has been produced by this tool, which uses the Graphviz open source package). The graphical visualization allowed us to manually check the result of the algorithm for a sample of diagrams, illustrating the various TERMOS constructs. Note that the prototype extracts the atoms from raw diagrams exported in the SVG (Scalar Vector Graphics) format. This allowed us to get quick feedback on the algorithm, as we did not bother specializing UML editors and parsers to a TERMOS profile. The solution is however not satisfactory for the long run.

The second prototype, developed at Budapest, aimed to demonstrate the integration of TERMOS into UML support technology. The tool was developed as an Eclipse plug-in, as the Eclipse platform has extensive built-in support to manipulate UML models. A TERMOS UML profile was developed to tag TERMOS scenarios and configurations with special stereotypes. The plug-in loads an Eclipse UML2 compliant model, and operates on the UML elements directly. It first searches for the elements with the TERMOS stereotypes, and can perform the syntactic and semantic checks described in Section 5. As the next step it generates the automaton for a TERMOS interaction, and stores it as an XML file. Finally, a test trace specified in a simple textual format can be checked against the automaton, and the trace is categorized as valid, invalid or inconclusive. Note, that the tool is not connected to *GraphSeq*, i.e. it requires that nodes in the traces should be mapped to nodes defined in the scenario manually.

7 Conclusion

This paper presented an approach that uses graphical scenarios, describing key interactions in mobile computing systems, for test and verification activities. The formal language TERMOS, based on UML Sequence Diagrams, is at the core of the approach. For space constraints, the presentation of TERMOS remained high-level. Its aim was to convey the main concepts underlying the language:

- TERMOS incorporates three elements we found useful to model scenarios in mobile settings. It has a spatial view, allowing us to abstract movement and node creation/shutdown by a sequence of labeled graphs. Its event view may contain configuration change events. Communication events include local broadcast.
- TERMOS is used to specify properties for subsets of nodes exhibiting predefined patterns of spatial configurations. The properties concern the partial orders of their communication and configuration change events. They come in various forms: positive requirements, negative requirements and test purposes.
- The TERMOS semantics combines graph matching and an operational state-based semantics for UML Sequence Diagram constructs.

The aim of TERMOS is the automated checking of test traces. We do not have yet a complete tool chain for this analysis, but the major pieces are there. The graph matching tool, developed at LAAS, has been presented in [4]. The production of the automaton was presented in this paper. Future work will build on the existing tools to integrate the two parts of the semantics. We will retain the principle of using UML support technology, demonstrated by the Budapest contribution.

Acknowledgement. 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.

References

1. ITU-T: Recommendation Z.120: Message Sequence Chart (MSC) (2004)
2. Object Management Group: UML 2.2 Superstructure Specification, formal/09-02-02. http://www.omg.org/docs/formal/09-02-02.pdf (2009)
3. Nguyen, M. D., Waeselynck, H., Rivière N.: Testing Mobile Computing Applications: Toward a Scenario Language and Tools, In: Int. Workshop on Dynamic Analysis (WODA 2008), pp 29-35, (2008)
4. Nguyen, M. D., Waeselynck, H., Rivière N.: GraphSeq: a Graph Matching Tool for the Extraction of Mobility Patterns, In: 3rd IEEE Int. Conf. on Software Testing, Verification and Validation (ICST 2010), IEEE CS Press, Paris, France (2010)
5. Waeselynck, H., et al.: Refined Design and Testing Framework, Methodology and Application Results, Hidenets D5.3. http://www.hidenets.aau.dk/Public+Deliverables (2008)
6. Baumeister, H., Koch, N., Kosiuczenko, P., Stevens, P., Wirsing, M.: UML for Global Computing. In: Global Computing, LNCS 2874, pp. 1-24. Springer-Verlag (2003).
7. Grassi, V., Mirandola, R., Sabetta, A.: A UML Profile to Model Mobile Systems. In: UML 2004, LNCS 3273, pp. 128-142. Springer-Verlag (2004)
8. Kusek, M., Jezic, G.: Extending UML Sequence Diagrams to Model Agent Mobility. In: 7th Int. Workshop on Agent-Oriented Software Engineering (AOSE 2006), LNCS 4405, pp. 51-63. Springer (2006)
9. Micskei, Z., Waeselynck, H.: The Many Meanings of UML 2 Sequence Diagrams: a Survey, Software and Systems Modeling, Online first, DOI:10.1007/s10270-010-0157-9, (2010)
10. Damm, W., Harel, D.: LSCs: Breathing Life into Message Sequence Charts. Form. Methods Syst. Des. 19(1), 45–80 (2001). DOI:10.1023/A:1011227529550
11. Sengupta, B., Cleaveland, R.: Triggered Message Sequence Charts. IEEE Trans. on Software Engineering, 32(8), pp. 587-607 (2006).
12. Klose, J.: Live Sequence Charts: a Graphical Formalism for the Specification of Communication Behavior. PhD thesis, C. v. O. Universitat Oldenburg (2003)
13. Küster-Filipe, J.: Modelling Concurrent Interactions. Theoretical Computer Science, 351(2):203–220 (2006)
14. Harel, D., Maoz, S.: Assert and negate revisited: Modal semantics for UML sequence diagrams. Software and Systems Modeling, 7(2):237–253 (2008).
15. Waeselynck, H., Micskei, Z., Nguyen, M. D., Rivière N.: Mobile Systems from a Validation Perspective: a Case Study, In: 6th Int. Symp. on Parallel and Distributed Computing (ISPDC 2007), IEEE Press, Hagenberg, Austria, July 5-8 (2007)
16. Huang, Q., Julien, C., Roman, G.: Relying on Safe Distance to Achieve Strong Partitionable Group Membership in Ad Hoc Networks, IEEE Trans. on Mobile Computing, 3(2), pp. 192-204, (2004)