A toolset for mobile systems testing

Pierre André, Nicolas Rivière, Hélène Waeselynck

▶ **To cite this version:**

Pierre André, Nicolas Rivière, Hélène Waeselynck. A toolset for mobile systems testing. 11th International Conference on Verification and Evaluation of Computer and Communication Systems (VEcOS 2017), Aug 2017, Montréal, Canada. pp.124-138. hal-01499518

HAL Id: hal-01499518
https://laas.hal.science/hal-01499518

Submitted on 31 Mar 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

Practical experience: tool description

A toolset for mobile systems testing

Pierre ANDRÉ*, Nicolas RIVIÈRE* and Hélène WAESELYNCK*
*LAAS-CNRS, Université de Toulouse, CNRS, UPS, Toulouse, France;
E-mail: {pierre.andre, nicolas.riviere, helene.waeselynck}@laas.fr

***Abstract*—Validation of mobile applications needs taking account of context (such network topology) and interactions between mobile nodes. Scenario-based approaches are well-suited to describe the behavior and interactions to observe in distributed systems. The difficulty to control accurately the execution context of such applications has led us to use passive testing. This paper presents a toolset which supports specification and verification of scenarios. A UML-based formal language, called TERMOS, has been implemented for specifying scenarios in mobile computing systems. These scenarios capture the key properties which are automatically checked on the traces, considering both the spatial configuration of nodes and their communication. We give an overview of the language design choices, its semantics and the implementation of the tool chain. The approach is demonstrated on a case study.**

***Index Terms*—Mobile computing systems, Scenario-based testing, UML sequence diagrams, UML profile, Trace analysis**

I. Introduction

Mobile computing systems involve devices (smartphone, laptop, intelligent car) that move within some physical areas, while being connected to networks by means of wireless links (Bluetooth, IEEE 802.11, LTE). Compared to "traditional" distributed systems, such systems run in an extremely dynamic context. The movement of devices yields an evolving topology of connection. Links with other mobile devices or with infrastructure nodes may be established or destroyed depending on the location. Moreover, mobile nodes may dynamically appear and disappear as devices are switched on and off, run out of power or go to standby mode.

Our work is aimed at developping a framework and a toolset to support the validation of such systems based on a passive testing approach. Passive testing (see e.g., [1]) is the process of detecting errors by passively observing the execution trace of a running system. In our case, the properties to be checked are specified using graphical interaction scenarios. A scenario captures a key property of a mobile application, depicting mandatory or forbidden behavior, to be checked on the traces. For mobile computing systems, a property has to be checked according to the topology of the mobile nodes involved in the scenario. A scenario should have both (i) a spatial view, depicting the dynamically changing topology of nodes as a sequence of graphs, and (ii) an event view representing the communications between nodes.

Graphical scenario languages (e.g., Message Sequence Charts [2], UML Sequence Diagrams [3]) allow the visual representation of interactions in distributed systems. Typical use cases, forbidden behaviors, test cases and many more aspects can be depicted. The need to automate analysis of test traces led us to design a scenario language with a formal semantics. This graphical language is a formal UML-based language called TERMOS (Test Requirement Language for Mobile Settings) [4]. It allows to depict scenarios including the node context.

The global concept followed for testing mobile computing applications needs two main elements: an execution platform and software tools to process the recorded data. The execution platform is composed of three elements: a context controller (to manage mobility of nodes), a network controller (to manage communication) and an execution environment support to run the system under test (SUT). The SUT is run in a simulated environment, using a synthetic workload. The platform allows to control the context, to observe and to record execution traces data during the testing campaigns.

Nevertheless, software tools are needed for checking a large amount of traces against scenarios. In order to support the verification, we designed a test framework. There are three main activities: the specification of scenarios, the capture of traces via the execution platform, and the analysis of traces.

In a previous work [5], we demonstrated our approach with a first prototype which integrated the TERMOS language and algorithms in an open-source UML environment. Since then, significant improvements and extensions have been achieved:

1) the different pieces of our toolset are fully integrated,
2) the specification of the language has been extended to take account of predicates,
3) severel checks have been implemented during specification and analysis,
4) a man-machine interface has been created for an accurate analysis of test verdicts.

This paper aims at giving an overview of a full demonstrator for the approach, from the graphical editing of requirement scenarios to their automated use for checking test traces.

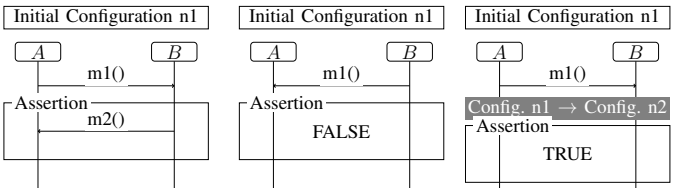
The structure of the paper is the following:

- section II gives an overview of the test framework.
- section III presents the choices and development done for the specification of graphical scenarios for mobile settings.
- section IV explains the principle of the trace analysis.
- section V concludes with results provided by the application of the test method to a case study.

II. TOOLSET OVERVIEW

The goal of our work is the validation of mobile computing systems. Our work focuses on the use of scenarios to analyze execution traces of mobile computing systems. It is essential to take account the network topology and the interactions between mobile nodes. To achieve this analysis, we need to run the application within an execution platform. It can be either a platform producing real traces from real physical devices or a simulation platform. For a better reproducibility of the test campaigns, the system under test (SUT) is run in a simulated environment, using a synthetic workload. The SUT may involve both fixed nodes and mobile devices. The movement of the latter ones is managed according to some mobility model, a context manager being in charge of producing location-based data. The network simulator can simulate delays or communication errors on wireless or wired links. Execution traces are collected, including both communication messages and location-based data from which the system spatial configurations can be retrieved.

We want to check whether the test trace exhibits some behavior patterns described by scenarios. The properties are specified using graphical interaction scenarios which represent test requirements or test purposes. The TERMOS language has been developed to capture the three classes of scenarios exemplified by Figure 1. *Positive requirements* capture key invariant properties of the following form: whenever a given interaction happens in the trace, then a specific interaction always follows. *Negative requirements* describe forbidden behaviors that should never occur in the trace. Any observed violation of a requirement must be reported. *Test purposes* describe behaviors to be covered by testing, that is, we would like these behaviors to occur at least once in the trace. If the interaction appears in the trace, the test purpose is reported as covered.



(a) Positive req. (b) Negative req. (c) Test purpose

Fig. 1. Requirement and test purpose scenarios (event views)

We interpret TERMOS scenarios as generic behavior patterns that may be matched by various subsets of the system during the test run. In Figure 1, the node ids *A* and *B* are symbolic node ids. For example, the positive requirement (Figure 1a) is interpreted as:

> Whenever two nodes exhibit spatial configuration *n1*, and the node matching *A* sends message *m1()* to the node matching *B*, then the node matching *B* must answer with message *m2()*.

At some point of a test run, we may have two simultaneous instances of *n1*, one with system nodes *x* and *y* matching *A* and *B*, and one with *x* and *z*. At some later point, system node *x* may play the role of *B* in yet another instance of *n1*.

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 using an automaton.

Assuming that system configuration graphs can be built from the contextual test data, step 1 can be formulated as a graph matching problem. We explained in [6] [7] how subgraph isomorphism can be used to search for all instances of the scenario configurations in a trace. Then, in step 2, the order of communication and configuration change events are analyzed using an automaton for all found spatial matches.

To automate the execution of the test and the processing of the traces, a test framework has been implemented. We made the choice to distribute the different steps in three main activities, as shown in Figure 2, each performing a specific task in the testing of mobile applications: the specification of scenarios, the capture of traces via the execution platform, and the analysis of traces.

The overall principles of the toolset are the following. The trace capture provides execution traces where location-based data and communication messages are time stamped. Requirement scenarios are specified manually within a UML-workshop. A scenario is transformed, after checks, into an automaton and a pattern containing a sequence of topologies. Finally, trace analysis is processed in fours steps with specific tools we have developed, and is concluded with a verdict(pass, fail, inconclusive).

III. SCENARIO SPECIFICATION

The scenario specification, the right green block in Figure 2, consists of three steps: **scenario modeling**, **scenario fomat checks** and **scenario transformation**.

In the first step, requirements are captured using our scenario based language, which includes the mobility related extensions. Our language TERMOS is a specialization of UML Sequence Diagrams [3]. Its genesis can be found in our work [8] [4]. Like in usual sequence diagrams, lifelines are drawn for the nodes and the partial orders of their communications are shown. We first noticed that the spatial configurations of nodes should be a first class concept. As a result, a scenario should have both (i) a spatial view, depicting the dynamically changing topology of nodes as a sequence of graphs, and (ii) an event view representing the communications between nodes. The syntax of the language includes elements for representing spatial configurations, changes in the communication structure, broadcast messages and predicates. These are called non standard elemetns in the following.

Next, the scenario to be verified automatically must meet a number of constraints. In this way, a verification stage for the well-formedness of the scenario has been set up. It aims at ensuring the correct syntactic form of the scenario, and

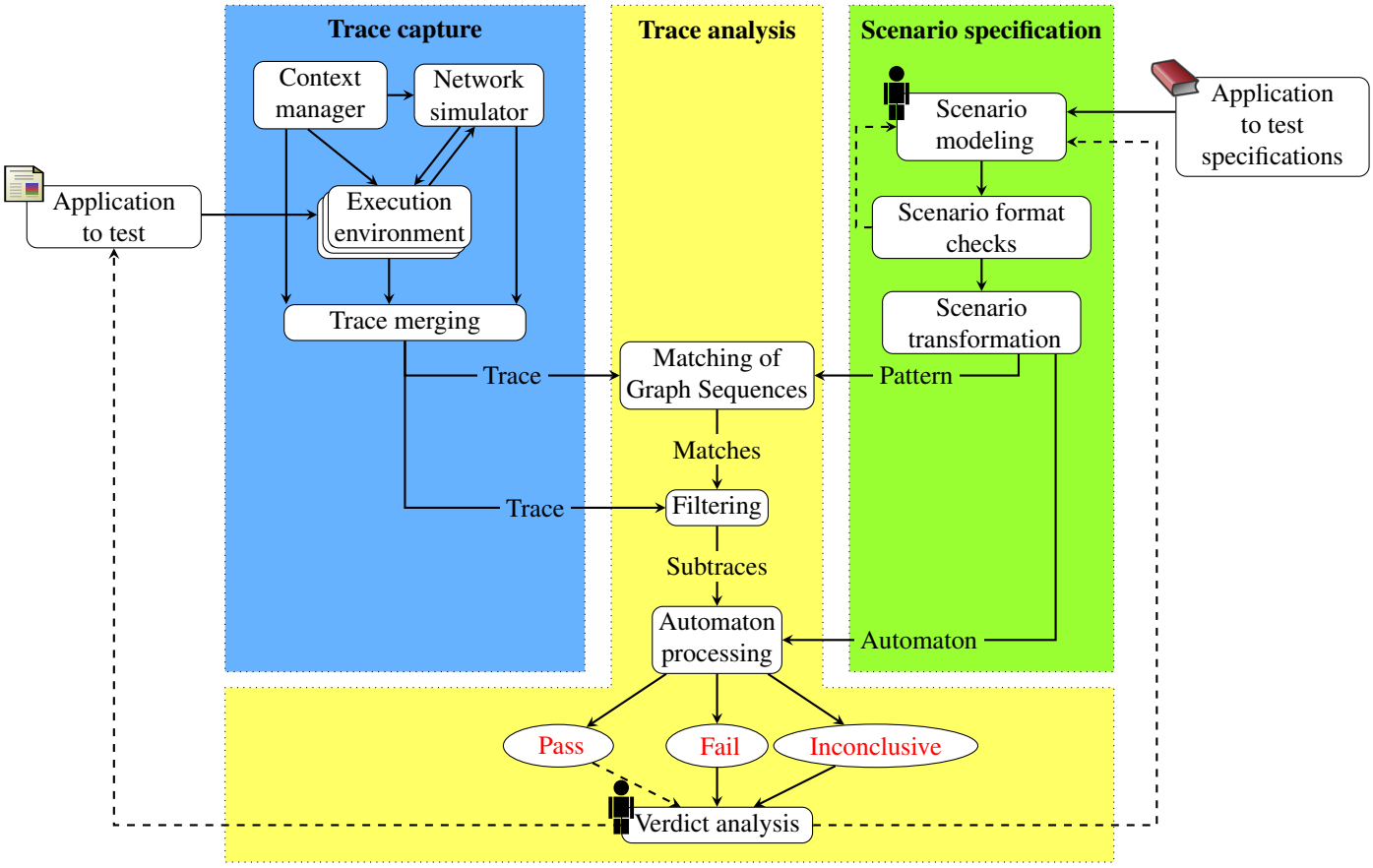


Fig. 2. Test framework architecture

the determinism of verdicts with the use of an unambiguous semantics. There are three categories of specific constraints for the language: *UML* syntactic restrictions, consistency between event views and spatial views, and specific elements of the language.

After all format checks passed successfully, it is possible to process the transformation of the graphical scenario in a suitable format for the trace analysis. The scenario is then decomposed in two files: a graph sequence representing the sequence of spatial configurations, and an automaton representing all event order paths available for this scenario.

Our language has been implemented within a *UML* workshop. For this implementation, we have chosen *Eclipse Papyrus* workshop through its extension possibilities through the use of *UML* profiles and the development of *Eclipse* plugins.

A. Scenario modeling

1) UML profile for non-standard elements: A scenario in a mobile setting contains two connected views. These views were integrated into Papyrus with the use of a UML profile in order to allow the representation of non-standard UML elements and some syntactic restrictions to sequence diagrams. The first two views are relevant to the mobile setting, while the syntactic restrictions are relevant to the use of TERMOS for checking execution traces. With this profile, we proposed three extensions: representation of a spatial view, consideration for spatial configuration change events in sequence diagrams, representation of broadcast communication events.

The sequence diagram illustrated Figure 3 depicts a piece of TERMOS scenario. The upper note "Initial Configuration: C1" reports that our scenario starts with a topology called *C1*. There is a life line in the diagram for each node of the spatial configuration. The spatial configuration change "CHANGE(C2)" impacts all the nodes, this is why this event is common to each life line. The "hello" broadcast message is sent by node *n2* and received by every node at communication range. This sequence diagram implies that nodes *n1*, *n3* and *n4* are connected with *n2*.

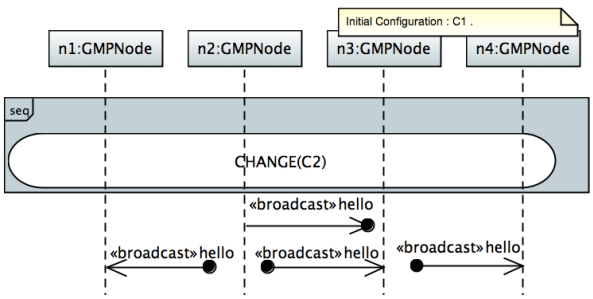


Fig. 3. TERMOS example

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.). In order to represent a broadcast message in the neighborhood, we used a stereotype $<< broadcast >>$ associated with an integer attribute to link several events together, as in Figure 4. This stereotype can be applied to lost/found message events as represented in Figure 3.

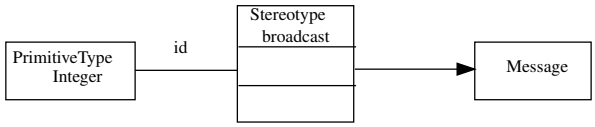


Fig. 4. UML profile for broadcast

2) A grammar for the predicates: In order to provide a richer description of scenarios, we extended the specification language to take account of predicates [9]. The scenario in Figure 5 contains two expressions to evaluate in the Assert block.

$$(m1.members\ includes\ n1)\ and\ (m1.members = m2.members)$$

Variables used in this expression may have various origins, e.g. variables from nodes as node identifier or node attributes, or message attributes from the event view. For example *n1* is a node identifier from the spatial view. Variable like *m1.members* comes from the content of the first message. The ability to use variables from either spatial and event view of a scenario is very useful and allows to represent behavior of complex systems in a scenario.

A dedicated grammar has been created to write predicates [9]. It is based on a subset of the *OCL* language syntax and has been implemented using the *ANTLR* language. The operations feasible using our grammar can be classified into three groups: numerical comparison, set comparison and logical operation. In the example Figure 5, we want to know if *n1* is a member of the *m1.members* list, and if *m1.members* and *m2.members* contain the same elements.

B. Scenario format checks

Before processing the verification steps of scenarios on an execution trace, it is necessary to verify that the scenario complies with the constraints of our language. The objectives of this stage are: to ensure the correct syntactic form of scenarios, and to ensure the determinism of verdicts using an unambiguous semantics.

Some checks are run before the scenario is transformed into an automaton. A feedback is provided to the user with all possible details allowing him to modify the scenarios. The goal is to have an error free scenario for the next steps. The constraints introduced by our language may be classified into three categories.

1) UML *syntax restrictions:* To adapt the sequence diagrams representation of interactions within mobile systems, some UML elements have been deleted and some constraints have been introduced [10]. For example, the following operators where deleted : *Strict*, *Loop*, *Ignore*, *Neg*, *Break* and *Critical*. In terms of constraints, some operators are considered as global events. The *Assert* operator in Figure 5 must cover all the lifelines of the scenario and be the last element of the sequence diagram.

2) Consistency constraints between event view and spatial view: As our scenarios are composed of two views, we must ensure consistency between them. For example, the nodes present in an event view must be present in the spatial view.

3) Specific elements of the language: The link between the spatial view and the event view of the scenario is managed by a global event called *Configuration Change*. An example of this specific element of the language is represented in Figure 3 by the event *CHANGE(C2)*. Another specific element is the broadcast communication event. Here we need to link several receive message events with one unique send event.

C. Scenarios processing

Before running trace analysis, a scenario transformation is mandatory to generate input patterns for trace analysis. This step occurs after all format checks have been passed successfully. As mentioned in the Figure 2, the scenario transformation process produces two behavior patterns. An event order analysis of the scenario is run to produce them. The configuration changes in the scenario are analyzed to build a sequence of graphs that depicts the needed sequence of network topology. An automaton is also built with all possible event sequences that may represent the scenario [4] [9].

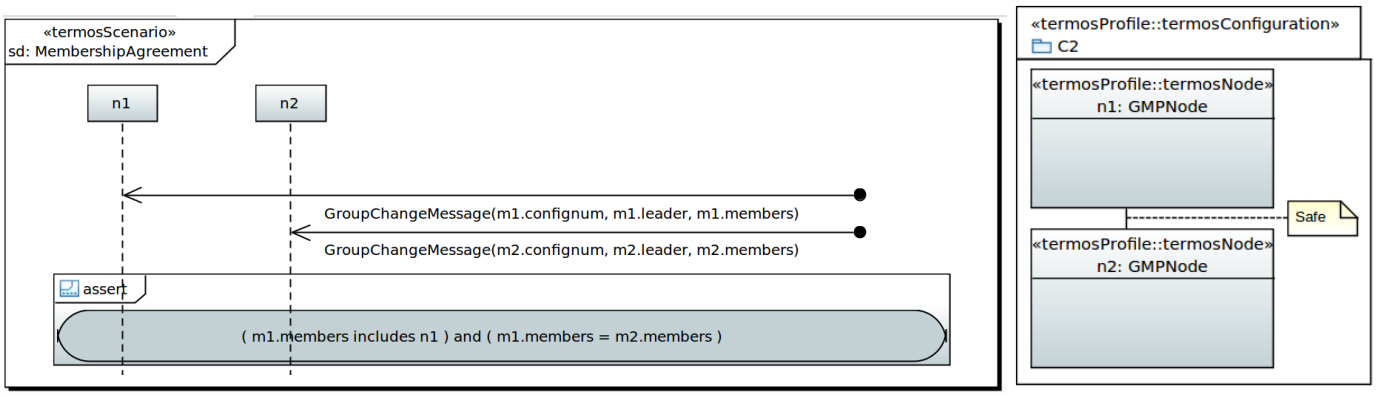
Some checks have to be executed once the complete automaton is produced. This is the case when a scenario includes some predicates. For each predicate, it is necessary to check that all the variables required for its assessment are available. Considering the example in Figure 5, the predicate in the Assert block uses variables *m1.members* and *m2.members* from the two preceding messages. All the branches that led to a state where a predicate is assessed must contain a valuation of the variables.

IV. Trace analysis

The trace analysis part (the central yellow block in Figure 2) consists of severals steps: **matching of graph sequences**, **trace filtering**, **automaton processing** and **verdict analysis**.

A. Principles

There is a gap between the abstract spatial configurations defined in the requirement scenarios and the concrete ones observed in the trace. It is necessary to decide which node from the execution environment can play the roles depicted in the scenarios. Based on their types and connections, the abstract nodes have to be mapped to the concrete ones found in the trace. However, usually there are several possible matchings. Moreover, the matching should take into account not only one configuration, but also the changes in a sequence of configurations like *C1* followed by *C2* in Figure 3. To address this issue, we developed a method and a tool, called



(a) Event view (b) Spatial view

Fig. 5. Scenario example using predicates

GraphSeq [4] [7], which reason on a series of abstract and concrete configuration graphs, and return the set of possible matches and valuations.

For each occurrence of the pattern identified by GraphSeq, a match is returned. A match identifies a subset of concrete nodes that exhibits the searched sequence of patterns and its duration. It contains the valuation of each variable presents in the pattern. For each match, a sub-trace is generated. A sub-trace contains only configuration changes identified by GraphSeq, and communication messages sent or received by the identified nodes during the time window.

All sub-traces are evaluated with respect to the requirements using the automaton. Each event in the sub-trace triggers a transition in the automaton until the check reaches a final state or the end of the sub-trace. Each state of the automaton is linked to a verdict. Finally, a pass, fail or inconclusive verdict is assigned to a *(trace, requirement, matching)* combination according to the reached state.

During the analysis of the execution trace, a new check has to be performed. Indeed, the use of predicates may lead to some cases of non-determinism. To detect them, from the current state, we are seeking if more than one transition can be fired.

Each sub-trace can lead to several verdicts. Indeed, in the automaton some transitions are labelled as initial ones. That means they can trigger a new validation of the automaton from the fired event. The example in Figure 6 is a trace we want to check against the scenario of 0Figure 5. Each message (from ❶ to ❺) leads to a verdict. On the other hand the occurence of ❻ does not lead to a verdict because it occurs after a configuration change. ❻ is represented here in order to explain how our tool works. However it cannot occur in a sub-trace because it will be filtered by the time window founds by GraphSeq in the previous steps of analysis.

B. Verdict analysis

The automaton execution stage leads mostly to a set of verdicts that it is difficult to analyze manually. We have implemented a verdict analysis tool based on a man-machine

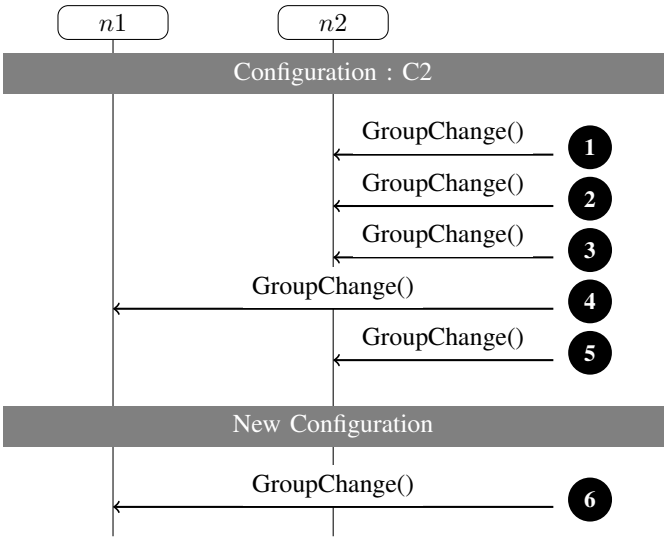


Fig. 6. Multiple verdicts from one trace

interface allowing an accurate analysis of test verdicts. The interface of this tool shown Figure 7 is composed of three areas. The first one contains all verdicts of the current execution. It may contain a large number of verdicts because an automaton check can be run on all the sub-traces identified in the first step of the trace analysis. In this part of the interface, the user is able to select and analyze in details a verdict through the analysis of the automaton transition triggered to reach the verdict. The trigerred transitions are displayed in the automaton area. For each transition, the event that triggered it is highlighted in the trace area. It is also possible to display the content of the message for manual analysis.

With this analysis interface, the user has a better feedback to understand more easily the behavior of the system and take the necessary actions to correct it. Actions can be changed in the source code of the application under test or the scenario may be rewrited in the right way. These actions are represented in Figure 2 by the feedback from **Verdict analysis** to

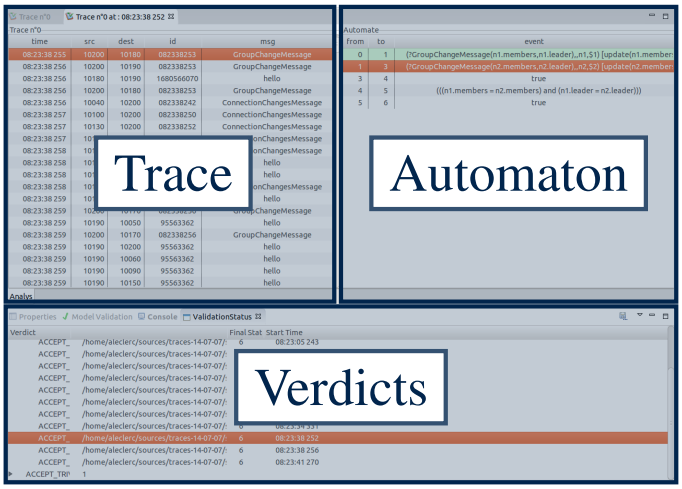


Fig. 7. Verdict analysis interface

Application to test and **Scenario modeling**.

V. Results

Our test framework was validated with a case study. To be used, our method requires two elements: detailed specifications in order to design test cases and functional implementation for the collection of execution traces. The application we have verified is a group membership protocol called GMP whose specifications are detailed in [11].

The main functionality of the GMP is to maintain a consistent view of who is in the group. The studied implementation dynamically merges and splits groups of nodes according to a notion of safe distance. The safe distance is determined to give enough time for two nodes in the same group for: (i) moving away from each other at their maximum speed, (ii) splitting up the group before loosing network connection between them. Using this notion, a link between two nodes can have three labels: *safe distance*, *communication range*, *disconnected* and *any* (wildcard for any of the three labels).

The GMP works with two main operations, *GroupSplit* and *GroupMerge*. We used specifications of the protocol fully described in [11] to write tests scenarios and check key properties of this protocol.

One example of property was described using our language in Figure 5. This scenario depicts a property called *Membership Agreement*. It aims at ensuring that two nodes connected together with a *safe* link have to be in the same group.

To collect traces of execution, we have instrumented the application under test. This instrumentation consists, on each mobile node, to simulate a location device acting as a GPS and to record all communications events with the outside world. To observe how the protocol works, several nodes must be running at the same time. To manage the simulation and coordinate nodes movements between them, a mobility simulator was used.

We tested our scenarios considering the traces collected during the execution of the GMP using 16 nodes during 15 minutes. This execution produced a global trace containing 900 configuration graphs and more than 500,000 sent or received message events. Firstly, as shown in Figure 2, for each scenario, we searched for spatial patterns and generated subtraces. This step found 3,116 spatial matches and generated one subtrace for each of them. Next, we check the automaton on each subtrace. The automaton may be checked more than once per subtrace because we look for *initial* events in the subtrace, and run the automaton starting at each initial event detected. This step led to 36,460 executions of the automaton, distributed as follows:

- **16 186** executions reached the final state of the automaton. In other words, the verdict of the execution is *accept*.
- **20 240** executions have not reached the assert block of the scenario. In this scenario, this is due to the lack of one of the two *GroupChange* messages.
- **34** executions stopped inside the assert block. These cases revealed violation of the property.

Complete results about tested scenarios are presented Table I. With these results displayed in the verdict analysis interface presented in the previous section, the user is able to analyse each verdict and detect which events caused it.

All the *GMP* properties were described using *TERMOS* languages, and verified on traces from the SUT presented above. Each property was violated at least once. The results show the usefulness of our tool which can detect properties violation easily on execution traces. The user only needs to describe properties using our UML editor, run analysis on execution traces and analyse the results given by our tool.

VI. Related works

We did not find any testing framework or tools like our toolset. The closest work we can find is a methodology for testing autonomous systems based on graphical scenarios [12].

Other works have investigated how to incorporate mobility into UML scenarios [13] [14] [15]. However, the focus was more on logical mobility (mobile computation) than on physical one (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 notations.

Having a formal semantics is crucial for our objective of automated analysis of traces. We had a thorough look at existing semantics for UML Sequence Diagrams [10]. We also looked at other scenario languages distinguishing potential and mandatory behavior. The most influential work for the TERMOS semantics was work on Live Sequence Charts (LSC) [16] [17], as well as work adapting LSC concepts into UML Sequence Diagrams [18] [19].

GraphSeq implements an algorithm to match sequences of configurations: the sequence of symbolic configurations of the scenario, and the sequence of concrete configurations traversed during SUT execution. To the best of our knowledge, this is an

TABLE I
SUMMARY OF SCENARIOS VALIDATION FOR A TRACE OF MORE THAN 500,000 EVENTS.

Tested scenario	**Matchs**		**Accept**		**Reject**
	Spatial	Event	Stringent	Trivial	Reject
Local monotonicity	16	3,608	3,478	16	114
Self inclusion	16	3,608	3,606	0	2
Membership change justification	16	3,608	3,495	16	97
Membership agreement	3,116	36,460	16,186	20,240	34
*Wrong split**	8768	53702	0	53098	604
*Concurrent merge**	2450	2487	0	2336	151
Concurrent split	162	569	52	387	130

original contribution. The comparison of sequences of graphs has been much less studied than the comparison of two graphs. The closest work we found is for the analysis of video images. In [20], the authors search for sequences of patterns (called pictorial queries) into a sequence of concrete graphs extracted from video images. Some differences with us are that their patterns do not involve label variables, and that there is at most one possibility for matching a pattern node with an image object.

VII. CONCLUSION

This paper presented a tooling support for TERMOS, a UML-based scenario language for the testing of mobile computing systems. Its formal scemantics allows an automated analysis of test traces. A grammar for predicates has been created to accommodate richer descriptions of scenarios. Several checks during the verification process from the depiction of requirements scenarios to trace analysis have been implemented to ensure the correctness of our method. A man-machine interface was integrated to help the user to analyze the test verdict. The full integration of our main tools, GraphSeq and TERMOS, led us to develop a complete tool chain for the automated checking of test traces. The integration has been done in UML workshop called Papyrus.

In order to validate our test framework, we led experiments on a group membership protocol. During this experimentation, we have proven the efficiency of our tools using them on traces of several hundred of thousands events. Detailed information on the testing tools and the tested scenarios of the GMP case study are available at https://www.laas.fr/projects/TERMOS. The tool will be made available online for the scientific community

REFERENCES

[1] A. Cavalli, S. Maag, and E. M. de Oca, "A passive conformance testing approach for a MANET routing protocol," in *Proceedings of the 2009 ACM symposium on Applied Computing*, ser. SAC '09. New York, NY, USA: ACM, 2009, pp. 207–211. [Online]. Available: http://doi.acm.org/10.1145/1529282.1529326

[2] International Telecommunication Union, *Message Sequence Chart (MSC)*, 2011, recommendation Z.120. [Online]. Available: http://www.itu.int/rec/T-REC-Z.120

[3] Object Management Group, *Unified Modeling Language (UML) 2.4.1 Superstructure Specification*, 2011, formal/2011-08-06.

[4] H. Waeselynck, Z. Micskei, N. Riviere, A. Hamvas, and I. Nitu, "TERMOS: a formal language for scenarios in mobile computing systems," in *Mobile and Ubiquitous Systems: Computing, Networking, and Services (MobiQuitous 2010)*, Sydney, Australia, 12 2010, pp. 285–296.

[5] P. Andre, H. Waeselynck, and N. Riviere, "A UML-based environment for test scenarios in mobile settings," in *Int. Conf. on Computer, Information, and Telecommunication Systems (CITS 2013)*. IEEE, 2013.

[6] M. D. Nguyen, H. Waeselynck, and N. Riviere, "GraphSeq: A graph matching tool for the extraction of mobility patterns," in *Software Testing, Verification and Validation (ICST), 3rd Int. Conf. on*, april 2010, pp. 195 –204.

[7] P. Andre, N. Riviere, and H. Waeselynck, "GraphSeq revisited: More efficient search for patterns in mobility traces," in *European Workshop on Dependable Computing*, ser. LNCS. Springer, 2013, vol. 7869, pp. 88–95.

[8] M. D. Nguyen, H. Waeselynck, and N. Riviere, "Testing mobile computing applications: toward a scenario language and tools," in *Proc. of the 2008 int. workshop on dynamic analysis*, ser. WODA '08. ACM, 2008, pp. 29–35.

[9] P. Andre, "Test of ubiquitous systems with explicit consideration of mobility," PhD [in french], UPS Toulouse, Nov. 2015. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01261593

[10] Z. Micskei and H. Waeselynck, "The many meanings of UML 2 sequence diagrams: a survey," *Software and Systems Modeling*, vol. 10, no. 4, pp. 489–514, 2011.

[11] Q. Huang, C. Julien, and G. Roman, "Relying on safe distance to achieve strong partitionable group membership in ad hoc networks," *IEEE Trans. Mobile Comput.*, vol. 3, no. 2, pp. 192–205, Apr. 2004.

[12] Z. Micskei, Z. Szatmári, J. Oláh, and I. Majzik, "A concept for testing robustness and safety of the context-aware behaviour of autonomous systems," in *Agent and Multi-Agent Systems. Technologies and Applications - 6th KES International Conference, KES-AMSTA 2012,Dubrovnik, Croatia, June 25-27, 2012. Proceedings*, ser. Lecture Notes in Computer Science, G. Jezic, M. Kusek, N. T. Nguyen, R. J. Howlett, and L. C. Jain, Eds., vol. 7327. Springer, 2012, pp. 504–513. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30947-2_55

[13] H. Baumeister, N. Koch, P. Kosiuczenko, P. Stevens, and M. Wirsing, "UML for global computing," in *Global Computing. Programming Environments, Languages, Security, and Analysis of Systems*, ser. LNCS, vol. 2874. Springer, 2003, pp. 1–24.

[14] V. Grassi, R. Mirandola, and A. Sabetta, "A UML profile to model mobile systems," in *UML 2004 - The Unified Modeling Language. Modelling Languages and Applications*, ser. LNCS, vol. 3273. Springer Berlin / Heidelberg, 2004, pp. 128–142.

[15] M. Kusek and G. Jezic, "Extending UML sequence diagrams to model agent mobility," in *Agent-Oriented Software Engineering VII*, ser. LNCS. Springer, 2007, vol. 4405, pp. 51–63.

[16] W. Damm and D. Harel, "LSCs: Breathing life into message sequence charts," *Formal Methods in System Design*, vol. 19, no. 1, pp. 45–80, Jul. 2001.

[17] J. Klose, "Live sequence charts: A graphical formalism for the specification of communication behavior," Ph.D. dissertation, Carl von Ossietzky Universitat Oldenburg, 2003.
[18] D. Harel and S. Maoz, "Assert and negate revisited: Modal semantics for UML sequence diagrams," *Software and Systems Modeling*, vol. 7, no. 2, pp. 237–252, 2008.
[19] J. Küster-Filipe, "Modelling concurrent interactions," *Theoretical Computer Science*, vol. 351, no. 2, pp. 203–220, Feb. 2006.
[20] K. Shearer, S. Venkatesh, and H. Bunke, "Video sequence matching via decision tree path following," *Pattern Recognition Letters*, vol. 22, no. 5, pp. 479–492, 2001.