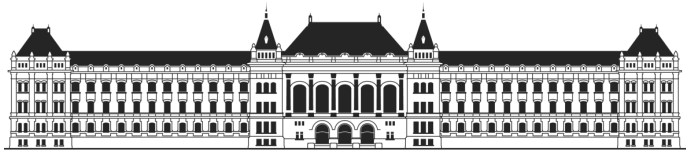


MŰEGYETEM 1782
Budapest University of Technology and Economics
Department of Measurement and Information Systems

Languages and Frameworks for Specifying Test Artifacts

PhD thesis booklet

Zoltán Micskei

Advisors:
István Majzik, PhD (BME)
Hélène Waeselynck, PhD (LAAS-CNRS)

Budapest, 2013

1 Preliminaries and objectives

Testing is an essential but complex and resource-consuming task in software development. IEEE defines testing as an "activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component" [IEE10]. Figure 1 depicts a high-level view of testing: test cases are created from the specification of the system, these are executed, and verdicts are assigned describing the outcome, e.g., passed or failed. Of course many important questions need to be answered before testing can be carried out. What part or functionality of the system needs to be tested? In what manner are the outcomes of the tests evaluated? Who decides whether a test passed or failed? To answer these questions several other *test artifacts* are needed besides test cases.

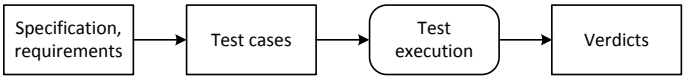


Figure 1: High-level view of the testing process

Figure 2 presents the artifacts used in testing in more detail[1]. From the high-level user requirements and the specification of the system *test requirements* are derived that define how the system should or should not behave in a certain situation. The *test approach* collects how and when the testing should be conducted, e.g., what processes, techniques, test levels and tools should be used. *Test purposes* describe what part or functionality of the system should be covered by testing. Later, *test case* specifications are created in which the inputs, predicted results, and set of execution conditions are specified. The expected output for a given input is obtained from a *test oracle*. The test cases are implemented, and with the help of *test adapters* they are executed in a *test execution environment*, which contains the *system under test* (SUT) and potentially some test doubles (drivers, stubs, etc.) that simulate the other components and the environment of the system. During the test execution *test traces* are recorded, which can contain the responses of the SUT, details about the changes in the test environment, etc. Finally, the outcomes of the test executions are evaluated, and *verdict*s are assigned. The set of possible verdicts is usually pass, fail, error (there was an error in the test execution environment itself), and inconclusive (neither a pass nor a fail verdict can be determined). These artifacts and the tools supporting them are combined into a *test framework*.

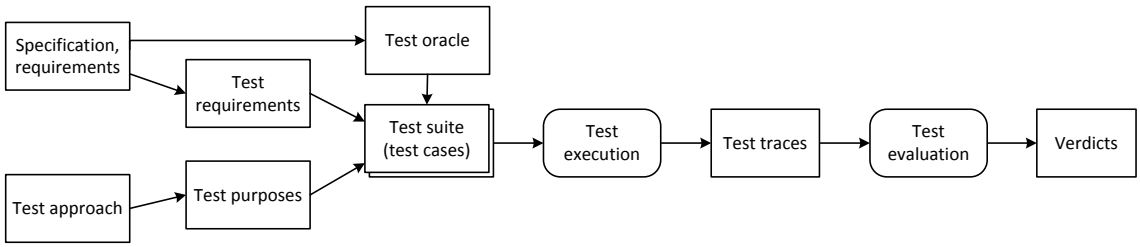


Figure 2: General test artifacts

Testing has an extensive literature (just to name a few few well-known books [Bei90; MS04]), numerous methods and techniques have been proposed to test different types of systems. In order to apply these methods, suitable languages are needed that can be used to precisely design and describe the above test artifacts.

[1] As testing is such a general term, these basic concepts have been defined in many ways. The dissertation follows mainly the IEEE terminology [IEE10] extended with the ISTQB glossary [IST10].

The research presented in this dissertation was focused on (i) what *languages* can be used to describe these test artifacts especially in application domains in which a proper solution is missing, and (ii) using the test artifacts how can *test frameworks* be constructed that can be used for testing in specific application domains.

1.1 Existing test languages and approaches

This section first gives examples of the existing languages used for describing test artifacts. Next, it presents how the Unified Modeling Language (UML), one of the most commonly used languages to model software systems, can be used in modeling test artifacts. Finally, different test approaches are introduced, which will be used in the dissertation.

1.1.1 Examples of languages for describing test artifacts

Depending on the type of the test artifact to describe, several methods and notations have been proposed. For example, test purposes can be described with labeled transition systems [JJ05] or with temporal logic formulae [Hon+01]. Test requirements can be extracted from UML models [BL02]. Test cases can be defined using TTCN-3 [ITU07], test configurations in the ATML language [IEE11]. Test oracles can be expressed as automata [Hes+08] or in SDL [Koc+98].

For describing partial behavior like test requirements or test purposes, a very common approach is to use graphical *scenario languages* [PJ04; KSH07]. They provide an intuitive yet powerful notation to express communication between different entities. Several language variants were proposed over the years. The International Telecommunication Union's (ITU) Message Sequence Chart (MSC) [ITU11] was one of the first of such languages. It is widely used, since its first introduction in 1993 it was updated several times. Live Sequence Chart (LSC) [DH01] concentrated on distinguishing possible and necessary behaviors. The dissertation focused on software systems, thus from the possible testing and modeling notations, the UML language was highly relevant.

1.1.2 Using UML 2 for specifying test artifacts

The Unified Modeling Language (UML) [OMG11b] developed by the Object Management Group (OMG) is one of the most commonly used languages to model software systems. UML has extensive tool support, and can be used in many aspects of software development from capturing requirements to specifying deployments. A recent paper by Cook [Coo12] presents a good overview of the history and evolution of the language.

To support the testing activities, a dedicated UML profile was developed. With the help of the *UML 2 Testing Profile* [OMG05] a UML model can specify (i) the test architecture, (ii) the behavior of test cases, and (iii) contents of the test data. The test architecture is modeled typically with stereotyped components for test context, arbiter etc. The behavior of test cases and test procedures are given usually with the scenario language found in UML, namely *Sequence Diagrams*, and the profile offers stereotypes to express default behavior or logging and validation actions. The test data stereotypes are used to define data partitions, and make expressing wildcards, omitted values possible.

The first version of Sequence Diagrams included in UML 1.x was similar to basic MSCs, i.e., it included lifelines representing communicating instances and messages going between lifelines. The next version introduced in UML 2.0 was a major rework; the language was extended with several complex, high-level elements. For example, new notations were added to express alternative or parallel flows. Moreover, what is even more significant from a testing perspective, language constructs were included to express mandatory and forbidden behavior, or messages

that can be ignored. However, the meaning of these elements, i.e. their *semantics*, was described only in natural language text fragments, which allowed several different interpretations.

Thus in order to use Sequence Diagrams to describe test artifacts or extend the language to cope with the characteristics of new application domains, first it should be identified what semantic variations exist for the language, and which of them fits for testing related activities.

1.1.3 Test approaches utilized in the dissertation

Testing activities can be differentiated based on what level they operate. Typical categories include module or unit testing (dealing with only one module), integration testing (checking the cooperation of several modules), and system testing (analyzing the whole system possibly taking into account its environment). The dissertation focuses on methods for *system testing*.

There are many approaches that can be applied at system level to test the functionality. One typical categorization, which is common in the protocol testing community, differentiates *active* and *passive testing* [AMN12]. In active testing the tests stimulate directly the SUT by providing inputs to it. However, this is not possible in some situations, e.g., when there is no direct interface to the SUT or the SUT operates in a complex environment. In these cases passive testing techniques offer an alternative, where the operation of the system is observed by recording *execution traces*, and then this trace is checked on-line or off-line to determine whether it conforms to the specification. This approach is common in testing distributed systems, where the test framework does not provide constant input to each of the nodes, instead it creates an initial test setup, and later observes the behavior of the nodes through their communication. In the application domains presented in this dissertation both active and passive testing were useful test approaches.

In system level testing usually not only the core functionality, but other *non-functional requirements* are considered. Non-functional requirements include performance or the different attributes of dependability [Avi+04], like robustness or availability. Such testing can be characterized with the following two components of the tests (stimuli); the *workload* triggers the (regular) operation of the system, while the *faultload* contains the different faults and stressful conditions applied on the system. Depending on how these two loads are balanced, different kinds of system properties can be tested, e.g., in API robustness testing only a faultload is executed against the public interfaces of the system, or in stress testing only a high level of workload is applied. One part of the research presented in this dissertation focused on robustness, which is the attribute of dependability that measures the behavior of the system under non-standard conditions. Robustness is defined by IEEE as "the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions" [IEE10].

As new application domains emerge and new system attributes become relevant to test, the classic methods have to be evaluated and one should identify whether new challenges have been arisen.

1.2 New application domains

The dissertation focuses on the following new and emerging application domains, which present several new challenges for the testing activities.

1.2.1 High availability middleware systems

Recently availability became a key factor even in common off-the-shelf computing platforms. High availability (HA) can be achieved by introducing manageable redundancy in the system.

The common mechanisms to manage redundancy and achieve minimal system outage can be implemented independently from the application in a component called a HA middleware. To standardize the functionality of such middleware systems leading IT companies formed the *Service Availability Forum* (SA Forum) to elaborate the *Application Interface Specification* (AIS) [SAF07]. Different vendors implemented the common specification in their solutions.

With multiple middleware products developed from the same specification the demand to compare the various implementations naturally arises. The most frequently examined properties are performance and functionality, but especially in case of HA products dependability is also an important property to be considered.

The characteristics of HA middleware systems can be summarized as follows.

State-based nature The complexity in testing these middleware implementations comes from the highly state-based nature of these systems: without a proper setup code most of the calls in the public interface result in trivial error messages, this way the valid operation cannot be tested. For example, the health of a component is checked in a callback, which needs to be registered after a connection to the middleware is initialized.

Robustness is a key factor As the availability requirements towards these systems are extremely high, HA middleware systems should handle even the unexpected situations. Typically testing effort is focused on all the valid paths and some of the common invalid inputs. However, in a HA middleware preparing for erroneous inputs is especially important, because an error in one component can render the whole system inaccessible if the middleware is not robust enough.

Testing, among other verification and validation techniques, can be used also to assess the robustness of a system. Specifically the goal of robustness testing is to activate robustness faults (typically design or programming faults) by supplying invalid inputs or presenting stressful environmental conditions.

Although robustness testing of HA middleware was a new research topic, previous robustness testing results from other application domains can serve as guidance. Early robustness testing experiments on command line programs generated a large number of random inputs [MFS90] (a technique known as “fuzzing”). As HA middleware systems have a common interface specification, a more relevant technique is *type-specific* testing introduced in the Ballista project [KDD08]. Here robustness tests were generated for POSIX compliant operating systems using valid and invalid values defined for the data types in the API. This method shall be extended to be used in testing the robustness of state-based middleware systems.

The challenge in describing tests for HA middleware systems lies in that (i) existing test languages mostly focus on conformance and not robustness, (ii) an HA middleware is a complex system that has several inputs that may trigger robustness faults, and (iii) its API is state-based with numerous functions, types and parameters.

1.2.2 Context-aware mobile computing systems

Mobile computing systems involve devices (handset, PDA, laptop, intelligent car, etc.) that move within some physical areas, while being connected to networks by means of wireless links (Bluetooth, IEEE 802.11, GPRS, etc.). Such devices represent an integral part of our life now. The specific characteristics of these systems can be summarized as follows.

Context awareness Context might be “any information that can be used to characterize the situation that are considered relevant to the interaction between a user and an application”

[BDR07]. For example, in mobile computing the context can be information collected by means of physical sensors such as location, time, speed of vehicle, or it can be information about network parameters, such as bandwidth, delay and connection topology. These systems have to take into account the actual context in their actions.

Dynamic, evolving environment In mobile computing systems the system structure, the number of mobile devices are not fixed. It varies over time, due to the dynamic appearance, suspension or stopping of nodes. Besides that, connectivity between nodes is also highly dynamic. As the nodes are free to move arbitrarily, they can join or leave the system in an unpredicted manner. Links may be established or destroyed, yielding an unstable connection topology.

Communication with unknown partners in local vicinity In ad hoc mobile networks, a natural communication is local broadcast. It is used as a basic step for the discovery layer in mobile applications (e.g., group discovery service for membership protocols, a route discovery in routing protocols, etc.). In this class of communication, a node broadcasts a message to its neighbors. As the topology of the system is unknown, the sending node does not know a priori the number and identity of potential receivers. Whoever is in transmission range of the sending node may listen and react to the message.

Existing languages presented in Section 1.1.1 were developed to describe mainly static configurations. Object creation or destruction can be depicted in some of the languages; however, these notations are not suitable to express frequent appearance or disappearance of other nodes or nearby objects. There were some works proposing extensions (e.g., [BM04; SE04]), but they concentrated mostly on mobile software agents and logical mobility. Moreover, existing languages focus on the communication between the entities, and do not offer an intuitive way to describe the actual context, i.e., the current state of the environment. Modeling language extensions and adaptation of existing test methods are needed that take into account the specificities of context-aware mobile systems.

1.3 Summarizing the new challenges

As the previous sections illustrated there are several relevant existing test approaches and languages, however they need to be adapted or extended to suit the new application domains. The following challenges summarize the *open research questions*, which have driven the work presented in the dissertation.

Challenge 1: Adapting robustness testing to HA middleware. How can relevant test inputs for a HA middeware be specified in test artifacts to support the automated testing of the robustness of such systems?

Challenge 2: Specifying mobile systems in test artifacts. How can dynamic, frequently changing communication structures and unknown partners be specified in test artifacts in a way that such systems can be later evaluated?

Instead of designing completely new test languages, we tried to reuse existing languages when possible. However, to incorporate new concepts into an existing language, that language should have a clear and precise semantics. From the available scenario languages, we focused on UML 2 Sequence Diagrams. But, as described previously, Sequence Diagrams can have several semantic

interpretations. Thus in order to define testing related extensions, first the semantics of UML 2 Sequence Diagrams has to be studied.

Challenge 3: Analyzing the semantics of UML 2 Sequence Diagrams. What semantic choices are available in UML 2 Sequence Diagrams, and what options can be chosen when the language is extended to support the description of test artifacts in a specific application domain?

Therefore the goal of the dissertation was to define test frameworks addressing these challenges, and develop the necessary languages for expressing the various test artifacts in the frameworks.

2 Research method and new results

According to [SS07], a research activity can be classified as *basic* ("research for the purpose of obtaining new knowledge") or *applied research* ("research seeking solutions to practical problems"). Moreover, *classical research* (using the scientific method of "formulate hypotheses then check or test these by means of experiments and observations") and *technology research* ("research for the purpose of producing new and better artefacts") can be differentiated. Technology research belongs usually to applied research, and it uses an iterative process: (i) starting with a problem analysis in which the potential needs for the new artifact are collected; (ii) the new artifact is constructed in an innovative way; (iii) the new artifact is evaluated against the initial needs.

The research presented in this dissertation can be categorized as applied, technology research, as its goal is to create better artifacts for solving practical problems. As the artifacts included new modeling languages, an important question was how modeling languages can be constructed.

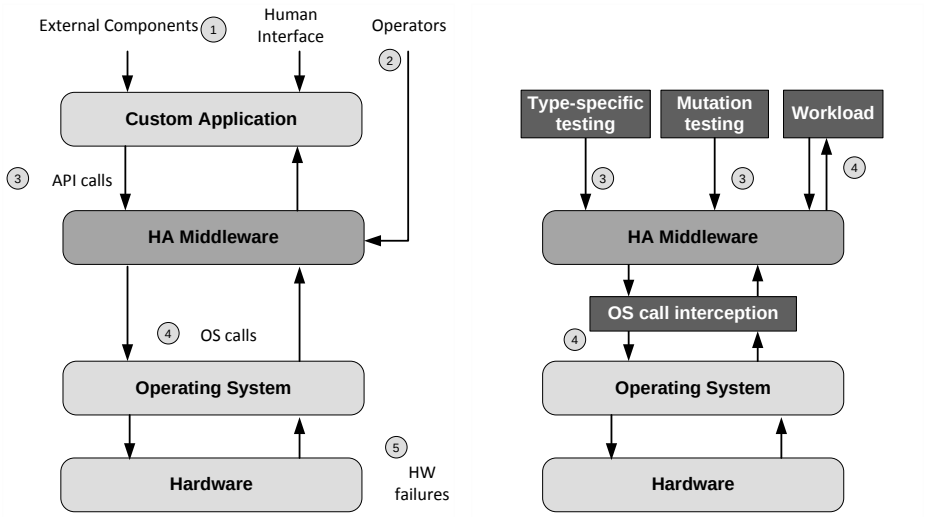
Language construction To solve the identified challenges, it was required to design new modeling languages and extend existing ones. Engineering a new language is a complex task, in order to precisely define a new modeling language the following artifacts have to be specified [OMG11a].

- *Abstract syntax* defines the main conceptual elements of the language and their relationships. The abstract syntax is meant for automated processing, and nowadays it is usually given using metamodels.
- *Concrete syntax* defines the human interface of the language (e.g., visual or textual notation). The elements of the concrete syntax have to be mapped to the elements of the abstract syntax.
- *Well-formedness rules* define additional constraints on the abstract syntax, which capture more complex conditions that cannot be specified otherwise easily in the abstract syntax.
- *Semantics* define the meaning of the language elements, usually with the help of a mapping to a well-defined semantic domain.

The rest of the section summarizes the new scientific results of the dissertation solving the challenges presented in Section 1.3.

2.1 Robustness testing of standard-based HA middleware

The first step of developing the test approach in the case of a "black box" AIS middleware was to identify the possible sources of inputs that can activate robustness faults. These inputs are depicted in Figure 3a, considering a typical computing node of a HA distributed system.



(a) Sources for activating robustness faults (b) Robustness test techniques

Figure 3: Overview of the test approach for HA middleware systems

The developed test approach focused on the following direct sources, as the thorough testing of the potential failures caused by the direct sources would cover a significant part of the failures induced by the other indirect sources.

The standardized middleware API calls are considered as a potential source of activating robustness faults as they represent faults in the applications, in external components used by the applications and human interaction also. The challenge in testing the API calls is that most of the AIS interface functions are state-based, i.e., a proper initialization call sequence, middleware configuration and test arrangement is required, otherwise a trivial error code is returned.

The failures of the OS system calls were included as they do not only represent the faults of the OS itself (which has lower probability for mature operating systems), but failures in other software components, in the underlying hardware and in the environment could also manifest in an error code returned by a system call. Possible examples of such conditions are writing data to a full disk, communication errors when sending a message, etc.

The following test approach was developed that utilizes a combination of three techniques to cover the selected sources, as depicted on Figure 3b.

- *Type-specific testing*: The robustness of the middleware's API functions should be tested in case of invalid values are used as parameters. This requires calling all the functions in the API of the middleware with a thorough combination of the possible valid and invalid values. The type-specific testing technique is used to construct such a robustness test suite, as this technique offers a systematic method to define the necessary valid and invalid test values.

- *Mutation-based sequential testing*: Some specific states of the middleware can only be reached by complex call sequences; nevertheless, the robustness of the API functions should

be tested even from these states. Thus first the middleware should be directed to these states, then its functions should be called with invalid inputs. The functional test suites provided by the vendors of the HA middleware could be used to reach these states as these test suites usually cover all the important states of the middleware. Therefore exceptional test sequences are constructed by using mutation operators on functional test suites that represent typical faults (e.g., changing the sequence of test calls, modifying parameters or function names).

- *OS call interception*: In order to test how the middleware reacts to the failures of the consumed lower-level services, the calls to the OS services should be intercepted and their return values should be modified. This interception can be realized with the help of a wrapper component that is placed between the middleware and the operating system libraries. Furthermore, this kind of test activity requires a workload application that drives the middleware in a way that the full range of the utilized OS calls could be observed and intercepted.

A robustness test framework was constructed for the above test techniques using the following systematic method:

1. First, the necessary test artifacts and their requirements were collected.
2. Next, the required test languages describing the test artifacts were constructed.
3. Finally, automatic tools were developed that can generate the test artifacts from descriptions given using the test languages.

The generated robustness test suite was executed on three different middleware implementations in several experiments to compare their robustness. The experiments identified several robustness failures in the implementations.

Thesis 1 *Following a systematic method I identified potential activation modes of robustness faults (including activation through stateless API, stateful API and underlying services), designed languages to represent the related test artifacts, and developed algorithms for tools that use these languages to generate test data. I implemented the languages and tools in a test framework, which can compare the robustness of standard specification-based middleware implementations.*

The work underlying Thesis 1 was a joint research with Francis Tam from Nokia Research Center, whose contributions included the overall directions of the investigation and discussions on the tool concept [Tam09].

The results of Thesis 1 are presented in Chapter 2 of the dissertation. Related publications are the following: [2], [5], [6], [8], [10], [11], [12].

2.2 Semantic choices in UML 2 Sequence Diagrams

When we first experimented with describing test requirements for the mobile system application domain using UML 2 Sequence Diagrams, we encountered the problem that the various formal semantics proposed for Sequence Diagrams handle even the most basic diagrams quite differently. It turned out that there are several subtle choices in the interpretation of language constructs. Moreover, these choices and all their consequences are often not obvious. We thus felt the need for a thorough review of the existing approaches before continuing with the test requirement language definition.

The following research approach was applied.

1. A literature review was conducted, and the proposed formal semantics for UML 2 Sequence Diagrams were analyzed.
2. The semantic choices faced by the existing approaches were collected and categorized.
3. The different options for the collected choices were evaluated.

As the above collection of choices and options can be useful for others extending or just simply using Sequence Diagrams for a special purpose, we searched for an easy to use presentation format. Usually the main barrier in using formal semantics is that understanding them requires substantial theoretical knowledge, thus we used a feature-model like representation [Kan+90] that is easily usable by engineers. Figure 4 depicts the options for one of the choices to illustrate the developed representation for the categorization. In this example *Categorizing traces* has two conflicting alternatives, and the *Two classes* choices can be optionally refined.

Table 1 presents the high-level choices identified to illustrate the types of decision one has to make when using scenario languages. The table includes choices about even the most basic constructs, e.g. how should a trace be represented, and choices coming up when dealing with advanced constructs like predicates in guards and invariants.

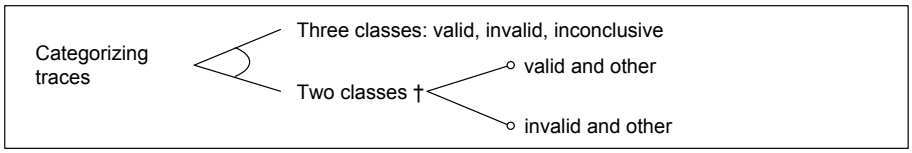


Figure 4: Options for the *Categorizing traces* choice

Thesis 2 *I identified and categorized the semantic choices and available options in UML 2 Sequence Diagrams. I gave a structured framework with an easy to use feature-model like representation of the available options that can be used to adapt the semantics of the language to a specific purpose.*

The results of Thesis 2 are presented in Chapter 3 of the dissertation. Related publications are the following: [1], [13].

2.3 A test language and framework for mobile systems

To better understand the new testing related challenges a case study [7] was performed, the analysis of a mobile Group Membership Protocol was carried out. The insights gained from this case study can be summarized as follows. Standard UML was appropriate to model the structure and behavior of one node (by static structure diagrams, state machines, activity diagrams etc.), but it was inconvenient for modeling a complex scenario which included several nodes, due to the lack of formal semantics assigned to sequence diagrams, lack of an unambiguous notation for broadcast messages etc. Moreover, services and applications in mobile settings rely heavily not just on user input but also on context information, like current location data. A test execution engine should be able to feed the System Under Test (SUT) not only with the messages coming outside from the SUT but also these contextual data.

Based on these studies a set of extensions were identified that were incorporated into UML 2 Sequence Diagrams to produce a language, called TERMOS (Test Requirement language for Mobile Setting), for describing test requirements for mobile systems. The language extension were carried out in the following way.

Table 1: Categorizing semantic choices in UML 2 Sequence Diagrams

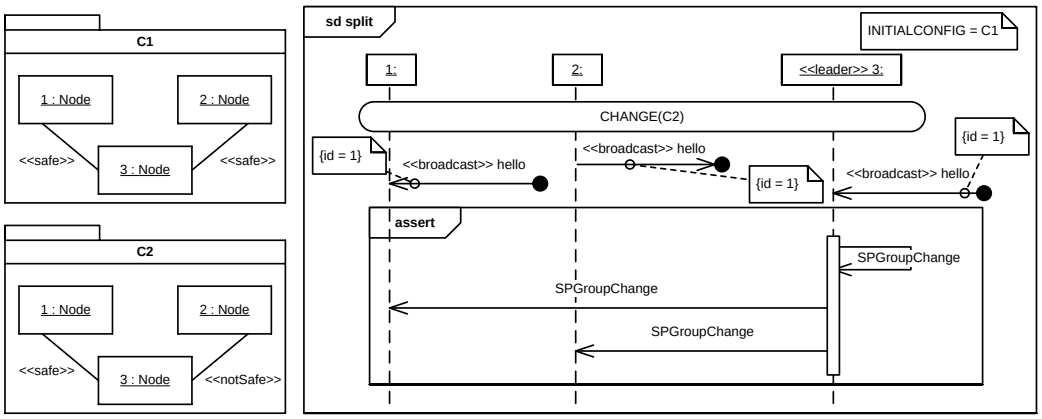
Interpretation of a basic Interaction	What is a trace? Categorizing traces Complete or partial traces
Introducing CombinedFragments	Combining fragments
Computing partial orders	Processing the diagram Underlying formalisms Choices and predicates
Introducing Gates	Gates on CombinedFragments Formal and actual Gates
Interpretation of conformance-related operators	Assert and Negate Ignore and Consider Conformance-related operators in complex diagrams Traces being both valid and invalid

- The abstract syntax of the languages was specified on one hand using the standardized way of defining a new UML profile with the appropriate stereotypes and tagged values, on the other hand by placing restrictions on the original metamodel of Sequence Diagrams to restrict the usage of certain elements.
- The concrete syntax used or reused the existing language elements. In this way the new test requirement scenarios can be modeled in existing UML tools, easing the usage of the new language.
- Further well-formedness constraints were defined to restrict the usage of certain combinations that would result in hard to check requirements, e.g., nesting of negated elements.
- The semantics was defined using the choices and options identified in Thesis 2. The formal semantics was inspired by the semantics defined for LSC [Klo03], which creates an automaton from a chart using a process called unwinding.

Figure 5 presents an example for a test requirement specified for a mobile system. Figure 5a contains the configuration fragments the scenario is referring to, while 5b captures the communication messages and time points, where the changes occur.

The testing framework requires a method to evaluate the requirements. Detailed traces are captured during testing containing the communication events and changes in the communication topology or the context of the system. A matching is then performed to search for nodes or context objects which could play the roles specified in the configuration or context fragment of the test requirement. Finally, based on the modalities defined by the conformance operators (assert, negate) in the scenario, the test requirement is evaluated and a verdict is attached to the test.

Thesis 3 *I designed a test requirement language that can be used in the domain of mobile systems. I defined the syntax of the language using extensions to the UML Sequence Diagrams' metamodel,*



(a) Spatial view (b) Event view

Figure 5: Example requirement scenario for a mobile system

and its semantics using an automaton-based formal operational semantics. The language is capable of expressing local broadcasts and changes in the communication topology, and has the necessary syntactic and semantic choices to make the specified requirements checkable.

Developing the test framework for mobile systems was a joint research with Nicolas Rivière and Minh Duc Nguyen from LAAS-CNRS. Designing and developing a tool called GraphSeq for matching parts of the test traces with test requirements was a contribution in Minh Duc Nguyen's PhD dissertation [Ngu09]. Áron Hamvas, an MSc student I supervised, created a tool [Ham10] for evaluating scenarios using the matching produced by GraphSeq.

The results of Thesis 3 are presented in Chapter 4 of the dissertation. Related publications are the following: [3], [4], [7], [9], [14].

3 Applications of new results

This section summarizes the practical applications of the results of the PhD dissertation.

3.1 Robustness comparison of AIS-based middleware implementations

The results presented in *Thesis 1* were applied in robustness testing and comparison of several HA middleware implementations.

- The robustness test suite was executed on three different middleware implementations: on OpenAIS [Opea], on OpenSAF [Opeb], and on Fujitsu-Siemens' SAFE4TRY. The results of the tests and comparisons of the different middleware implementations were published in [5], [6], [8].
- The faults identified in OpenAIS were reported to the open source community, and the whole robustness test suite was made publicly available [BME07].
- The robustness testing results for OpenAIS and OpenSAF were uploaded to the public AMBER Data Repository (ADR) [AMM10]. ADR is an open repository for uploading, analyzing and sharing benchmark and measurement data. Our robustness testing results can be downloaded or analyzed in ADR.

3.2 Testing mobile systems

The categorization of semantic choices in UML 2 Sequence Diagrams presented in *Thesis 2* was used to define a new testing language for mobile systems called TERMOS [4].

The TERMOS language (*Thesis 3*) was defined in the context of the HIDENETS EU FP6 research project [HID09]. The HIDENETS project (Highly dependable IP-based networks and services) developed and analyzed end-to-end resilience solutions for distributed applications and mobility-aware services in car-to-car communication scenarios with infrastructure service support.

These results are also used in the context of the ARTEMIS R3-COP research project [R3C11] to design a test requirement scenario language for autonomous systems. The R3-COP project (Resilient Reasoning Robotic Co-operating Systems) aims to develop new methodology and technologies to enable production of advanced robust and safe cognitive, reasoning autonomous and co-operative robotic systems in different application domains.

4 Publication list

Number of peer-reviewed publications:	16
Number of independent citations:	33

4.1 Publications related to the theses

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Thesis 1:		•			•	•		•		•	•	•		
Thesis 2:	•												•	
Thesis 3:			•	•			•		•					•

Journal paper

[1] Z. Micskei and H. Waeselynck. “The many meanings of UML 2 Sequence Diagrams: a survey”. In: *Software and Systems Modeling* 10.4 (2011), pp. 489–514. DOI: 10.1007/s10270-010-0157-9

Chapter in edited book

[2] Z. Micskei, H. Madeira, A. Avritzer, I. Majzik, M. Vieira, and N. Antunes. “Robustness Testing Techniques and Tools”. In: *Resilience Assessment and Evaluation of Computing Systems*. Ed. by K. Wolter, A. Avritzer, M. Vieira, and A. van Moorsel. Springer, 2012, pp. 323–339. DOI: 10.1007/978-3-642-29032-9_16

The section giving a survey on robustness testing was my contribution in the chapter.

[3] G. Pintér, Z. Micskei, A. Kövi, Z. Égel, I. Kocsis, G. Huszerl, and A. Pataricza. “Model-Based Approaches for Dependability in Ad-Hoc Mobile Networks and Services”. In: *Architecting Dependable Systems V*. Springer, 2008, pp. 150–174. DOI: 10.1007/978-3-540-85571-2_7

The section describing the testing method of mobile systems was my contribution.

International conference

[4] H. Waeselynck, Z. Micskei, N. Rivière, Á. 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)*. Ed. by P. Sénac, M. Ott, and A. Seneviratne. Sydney, Australia, Dec. 2010, pp. 285–296. DOI: 10.1007/978-3-642-29154-8_24

The concept of a separate event and spatial view, moreover the test framework using such scenarios was a joint contribution. My contributions included designing the syntax and semantics of the TERMOS language.

[5] A. Kövi and Z. Micskei. "Robustness Testing of Standard Specifications-Based HA Middleware". In: *30th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, June 2010, pp. 302–306. DOI: 10.1109/ICDCSW.2010.73

The robustness testing framework (test approach and tools) was my contribution, and András Kövi prepared, executed and analyzed the experiments on the OpenSAF middleware.

[6] Z. Micskei, I. Majzik, and F. Tam. "Comparing Robustness of AIS-Based Middleware Implementations". In: *International Service Availability Symposium (ISAS 2007)*. Ed. by M. Malek, M. Reitenspieß, and A. van Moorsel. Vol. 4526. LNCS. Springer, May 2007, pp. 20–30. DOI: 10.1007/978-3-540-72736-1_3

The concept of the robustness test framework was a joint contribution. Implementing the test tools and running and analysing the test experiments was my contribution.

[7] H. Waeselynck, Z. Micskei, M. D. Nguyen, and N. Rivière. "Mobile Systems from a Validation Perspective: a Case Study". In: *Sixth International Symposium on Parallel and Distributed Computing (ISPDC '07)*. Ed. by D. Kranzlmüller, W. Schreiner, and J. Volkert. Hagenberg, Austria: IEEE Computer Society, July 2007, pp. 85–92. DOI: 10.1109/ISPDC.2007.37

The static review of the specification and the implementation was my contribution in the paper. Minh Duc Nguyen performed the test experiments on the GMP case study.

[8] Z. Micskei, I. Majzik, and F. Tam. "Robustness Testing Techniques for High Availability Middleware Solutions". In: *International Workshop on Engineering of Fault Tolerant Systems (EFTS2006)*. Luxembourg, Luxembourg: University of Luxembourg, June 2006, pp. 55–66

The concept of the robustness test framework was a joint contribution. Designing and implementing the individual test tools was my contribution.

Local conference

[9] Z. Micskei. "Specifying Tests for Ad-Hoc Mobile Systems". In: *15th PhD Mini-Symposium*. Budapest University of Technology and Economics. Budapest, Hungary: Department of Measurement and Information Systems, Feb. 2008, pp. 32–35

[10] Z. Micskei. "Robustness Comparison of High Availability Middleware Systems". In: *14th PhD Mini-Symposium*. Budapest University of Technology and Economics. Budapest, Hungary: Department of Measurement and Information Systems, Feb. 2007, pp. 70–73

[11] Z. Micskei. "Robustness Testing of High Availability Middleware Solutions". In: *13th PhD Mini-Symposium*. Budapest University of Technology and Economics. Budapest, Hungary: Department of Measurement and Information Systems, Feb. 2006, pp. 36–37

[12] Z. Micskei. "Nagy Rendelkezésre Állást Biztosító Köztes Rétegek Robosztusság Tesztelése". In: *Proceedings of Tavaszi Szél 2006*. Kaposvár, Hungary: Doktoranduszok Országos Szövetsége, May 2006, pp. 295–298

Technical report

[13] Z. Micskei and H. Waeselynck. *A survey of UML 2.0 sequence diagrams' semantics*. Tech. rep. 08389. Laboratoire d'Analyse et d'Architecture des Systemes (LAAS), Aug. 2008, pp. 1–37

[14] Z. Micskei, H. Waeselynck, M. D. Nguyen, and N. Rivière. *Analysis of a group membership protocol for Ad-hoc networks*. Tech. rep. 06797. Laboratoire d'Analyse et d'Architecture des Systemes (LAAS), Nov. 2006, pp. 1–42

The insights gained from the case study was a joint contribution. I performed the specification review and reverse engineering the implementation.

4.2 Additional publications

Journal paper

[15] I. Kocsis, A. Pataricza, Z. Micskei, A. Kövi, and Z. Kocsis. "Analytics of Resource Transients in Cloud Based Applications". In: *Int. Journal of Cloud Computing* (). To appear. URL: http://www.inderscience.com/info/ingeneral/forthcoming.php?jcode=ijcc

[16] G. Pintér, Z. Micskei, and I. Majzik. "Supporting design and development of safety critical applications by model based tools". In: *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös nominatae Sectio Computatorica* XXX (2009), pp. 61–78

International conference

[17] 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*. Ed. by G. Jezic, M. Kusek, N.-T. Nguyen, R. Howlett, and L. Jain. Vol. 7327. LNCS. Springer, June 2012, pp. 504–513. DOI: 10.1007/978-3-642-30947-2_55

[18] I. Kocsis, A. Pataricza, Z. Micskei, I. Szombath, A. Kövi, and Z. Kocsis. "Cloud Based Analytics for Cloud Based Applications". In: *1st International IBM Cloud Academy Conference*. Research Triangle Park, USA, Apr. 2012, pp. 1–23

[19] F. Bouquet, R. Breu, J. Jurjens, F. Massacci, V. Meduri, Z. Micskei, F. Piessens, K. Stolen, and D. Varró. "SecureChange: Security Engineering for Lifelong Evolvable Systems". In: *European Future Technologies Conference and Exhibition (FET09)*. Poster Session. Prague, Czech Republic, Apr. 2009, pp. 101–102

[20] L. Gönczy, I. Majzik, A. Horváth, D. Varró, A. Balogh, Z. Micskei, and A. Pataricza. "Tool Support for Engineering Certifiable Software". In: *Electronic Notes in Theoretical Computer Science* 238.4 (2009). Proc. of 1st Workshop on Certification of Safety-Critical Software Controlled Systems (SafeCert 2008), pp. 79–85. DOI: 10.1016/j.entcs.2009.09.008

[21] I. Majzik, Z. Micskei, and G. Pintér. "Development of Model Based Tools to Support the Design of Railway Control Applications". In: *Computer Safety, Reliability, and Security*. Ed. by F. Saglietti and N. Oster. Vol. 4680. LNCS. Springer Berlin / Heidelberg, Sept. 2007, pp. 430–435. DOI: 10.1007/978-3-540-75101-4_41

[22] G. Pintér, Z. Micskei, and I. Majzik. "Supporting Design and Development of Safety Critical Applications by Model Based Tools". In: *10th Symposium on Programming Languages and Software Tools (SPLST 2007)*. Ed. by Z. Horváth, L. Kozma, and V. Zsók. Dobogókő, Hungary: Eotvos University Press, June 2007, pp. 61–75

[23] Z. Micskei and I. Majzik. "Model-based Automatic Test Generation for Event-Driven Embedded Systems using Model Checkers". In: *Dependability of Computer Systems (DepCoS-RELCOMEX 2006)*. IEEE Computer Society, May 2006, pp. 191–198. DOI: 10.1109/DEPCOS-RELCOMEX.2006.37

Local event

[24] Z. Micskei. "Automatikus tesztgenerálás modell ellenőrzővel". In: *X. Fiatal Műszakiak Tudományos Ülésszaka (FMTU)*. ed. by E. Bitay. Erdélyi Múzeum Egyesület. Kolozsvár, Románia, Mar. 2005, pp. 47–50

References

[AMM10] R. Almeida, N. Mendes, and H. Madeira. "Sharing Experimental and Field Data: The AMBER Raw Data Repository Experience". In: *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*. 2010, pp. 313–320. DOI: 10.1109/ICDCSW.2010.75.

[AMN12] C. Andrés, M. G. Merayo, and M. Núñez. "Formal passive testing of timed systems: theory and tools". In: *Software Testing, Verification and Reliability* (2012). DOI: 10.1002/stvr.1464.

[Avi+04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. "Basic Concepts and Taxonomy of Dependable and Secure Computing". In: *IEEE Trans. Dependable Secur. Comput.* 1 (2004), pp. 11–33. DOI: 10.1109/TDSC.2004.2.

[BDR07] M. Baldauf, S. Dustdar, and F. Rosenberg. "A survey on context-aware systems". In: *Int. J. Ad Hoc Ubiquitous Comput.* 2.4 (2007), pp. 263–277. DOI: 10.1504/IJAHUC.2007.014070.

[Bei90] B. Beizer. *Software Testing Techniques*. 2nd Edition. International Thomson Press, 1990.

[BL02] L. Briand and Y. Labiche. "A UML-Based Approach to System Testing". In: *Software and Systems Modeling* 1.1 (2002), pp. 10–42. DOI: 10.1007/s10270-002-0004-8.

[BM04] E. Belloni and C. Marcos. "MAM-UML: An UML Profile for the Modeling of Mobile-Agent Applications". In: *Chilean Computer Science Society, International Conference of the*. 2004, pp. 3–13. DOI: 10.1109/QEST.2004.14.

[BME07] BME. *Robustness test suite for OpenAIS framework*. 2007. URL: http://mit.bme.hu/~micskeiz/pages/robustness_testing.html#aisrobustness.

[Coo12] S. Cook. "Looking back at UML". In: *Software and Systems Modeling* (2012), pp. 1–10. DOI: 10.1007/s10270-012-0256-x.

[DH01] W. Damm and D. Harel. "LSCs: Breathing Life into Message Sequence Charts". In: *Formal Methods in System Design* 19.1 (2001), pp. 45–80. DOI: 10.1023/A:1011227529550.

[Ham10] Á. Hamvas. "Using UML Sequence Diagrams for the Requirement Analysis of Mobile Distributed Systems". MSc thesis. Budapest University of Technology and Economics (BME), 2010.

[Hes+08] A. Hessel, K. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou. "Testing Real-Time Systems Using UPPAAL". In: *Formal Methods and Testing*. Vol. 4949. LNCS. 2008, pp. 77–117. DOI: 10.1007/978-3-540-78917-8_3.

[HID09] HIDENETS. *Highly dependable ip-based networks and services*. EU FP6 Specific Targeted Research Project (STREP), IST 026979. 2009. URL: http://www.hidenets.aau.dk.

[Hon+01] H. S. Hong, I. Lee, O. Sokolsky, and S. D. Cha. "Automatic Test Generation from Statecharts Using Model Checking". In: *In Proceedings of FATES'01, Workshop on Formal Approaches to Testing of Software, volume NS-01-4 of BRICS Notes Series*. 2001, pp. 15–30.

[IEE10] Institute of Electrical and Electronics Engineers. *Systems and software engineering – Vocabulary*. Standard 24765:2010. 2010, pp. 1–418. DOI: 10.1109/IEEESTD.2010.5733835.

[IEE11] Institute of Electrical and Electronics Engineers. *IEEE Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML*. Standard 1671-2010. 2011, pp. 1–388. DOI: 10.1109/IEEESTD.2011.5706290.

[IST10] International Software Testing Qualifications Board. *Standard glossary of terms used in Software Testing*. Version 2.1. 2010. URL: http://istqb.org/display/ISTQB/Downloads.

[ITU07] International Telecommunication Union. *Testing and Test Control Notation version 3: TTCN-3 core language*. Recommendation Z.161. 2007. URL: http://www.itu.int/rec/T-REC-Z.161.

[ITU11] International Telecommunication Union. *Message Sequence Chart (MSC)*. Recommendation Z.120. 2011. URL: http://www.itu.int/rec/T-REC-Z.120.

[JJ05] C. Jard and T. Jéron. "TGV:, theory, principles and algorithms". In: *International Journal on Software Tools for Technology Transfer (STTT)* 7.4 (2005), pp. 297–315. DOI: 10.1007/s10009-004-0153-x.

[Kan+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Tech. rep. CMU/SEI-90-TR-021. Carnegie-Mellon University Software Engineering Institute, 1990.

[KDD08] P. Koopman, K. Devale, and J. Devale. "Interface Robustness Testing: Experience and Lessons Learned from the Ballista Project". In: *Dependability Benchmarking for Computer Systems*. Ed. by K. Kanoun and L. Spainhower. John Wiley & Sons, Inc., 2008, pp. 201–226. DOI: 10.1002/9780470370506.ch11.

[Klo03] J. Klose. "Live Sequence Charts: A Graphical Formalism for the Specification of Communication Behavior". PhD thesis. Carl von Ossietzky Universitat Oldenburg, 2003.

[Koc+98] B. Koch, J. Grabowski, D. Hogrefe, and M. Schmitt. "Autolink-a tool for automatic test generation from SDL, specifications". In: *Industrial Strength Formal Specification Techniques, 1998. Proceedings. 2nd IEEE, Workshop on.* 1998, pp. 114–125. DOI: 10.1109/WIFT.1998.766305.

[KSH07] H. Kugler, M. J. Stern, and E. J. A. Hubbard. "Testing scenario-based models". In: *Proceedings of the 10th international conference on Fundamental approaches to software engineering.* (Braga, Portugal). FASE'07. 2007, pp. 306–320. DOI: 10.1007/978-3-540-71289-3_24.

[MFS90] B. P. Miller, L. Fredriksen, and B. So. "An empirical study of the reliability of UNIX utilities". In: *Commun. ACM* 33.12 (1990), pp. 32–44. DOI: 10.1145/96267.96279.

[MS04] G. J. Myers and C. Sandler. *The Art of Software Testing.* John Wiley & Sons, 2004.

[Ngu09] M. D. Nguyen. "Méthodologie de test de systèmes mobiles : Une approche basée sur les sénarios". PhD thesis. Université Paul Sabatier - Toulouse III, 2009.

[OMG05] Object Management Group. *UML Testing Profile v1.0 (U2TP).* 2005. URL: http://www.omg.org/technology/documents/formal/test_profile.htm.

[OMG11a] Object Management Group. *Unified Modeling Language (UML) 2.4.1 Infrastructure Specification.* formal/2011-08-05. 2011.

[OMG11b] Object Management Group. *Unified Modeling Language (UML) 2.4.1 Superstructure Specification.* formal/2011-08-06. 2011.

[Opea] OpenAIS. *OpenAIS Standards Based Cluster Framework.* URL: http://www.openais.org.

[Opeb] OpenSAF. *OpenSAF Open Service Availability Framework.* URL: http://www.opensaf.org.

[PJ04] S. Pickin and J.-M. Jézéquel. "Using UML Sequence Diagrams as the Basis for a Formal Test Description Language". In: *Integrated Formal Methods.* Vol. 2999. LNCS. 2004, pp. 481–500. DOI: 10.1007/978-3-540-24756-2_26.

[R3C11] R3-COP. *Resilient Reasoning Robotic Co-operating Systems.* ARTEMIS research project nr. 100233. 2011. URL: http://www.r3-cop.eu/.

[SAF07] Service Availability Forum (SA Forum). *Application Interface Specification (AIS).* 2007. URL: http://www.saforum.org.

[SE04] K. Saleh and C. El-Morr. "M-UML: an extension to UML for the modeling of mobile agent-based software systems". In: *Information and Software Technology* 46.4 (2004), pp. 219–227. DOI: 10.1016/j.infsof.2003.07.004.

[SS07] I. Solheim and K. Stølen. *Technology research explained.* Tech. rep. SINTEF A313. SINTEF ICT, 2007.

[Tam09] F. Tam. "Service Availability Standards for Carrier-Grade Platforms: Creation and Deployment in Mobile Networks". Tampere University of Technology, 2009. URL: http://URN.fi/URN:NBN:fi:tty-200904281053.