Simulation and Testing of Mobile Computing Systems using Fujaba

Ping Guo
International Graduate School of Dynamic Intelligent Systems
University of Paderborn, Germany
ping@upb.de

Reiko Heckel*
Department of Computer Science
University of Dortmund, Germany
(on leave from University of Paderborn)
reiko@upb.de

ABSTRACT
The paper presents an approach for analysis, modeling and validation of mobile information systems with the tool support of Fujaba. The approach is developed based on UML-like meta models and graph transformation techniques to support sound methodological principals, formal analysis and refinement. With conceptual and concrete level of modeling and simulation, the approach could support application development and the development of new mobile platforms. The approach also provides automatic analysis, validation and behavior consistency check with the support of Fujaba.

1. INTRODUCTION
Mobility is a "total meltdown" of the stability assumed by distributed systems as stated in [7]. The main difference is caused by the possibility of roaming and wireless connection. Roaming implies that, since devices can move to different locations, their computational context (network access, services, permissions, etc.) may change, and the mobile hosts are resource limited. Wireless connections are generally less reliable, more expensive, and provide smaller bandwidth, and they come in a variety of different technologies and protocols. All these result in a very dynamic software architecture, where configurations and interactions have to be adapted to the changing context and relative location of applications.

Mobility has created additional complexity for computation and coordination, which makes the current architectural concepts and techniques hard to use [2]. The current architectural approach offers only a logical view of change; it does not take the properties of the "physical" distribution topology of locations and communications into account. It relies on the assumption that the computation performed by individual components is relative to location of the component, and the coordination mechanisms through connectors can be always transmitted successfully by the underlying communication network. In order to support mobility, the architectural approach needs to be adjusted in different abstract layers of modeling and specification languages.

As shown in [5], there are a lot of platforms and middleware have been developed for mobile computing. These different platforms and middleware provide different transparency levels of context awareness to the application, where the application has to be aware of, and be able to react to, changes in its context given by its current location, quality, cost and types of available connections, etc. The amount of context information required and available to the application greatly varies, depending on the employed infrastructure so that, in the end, not every intended application scenario may have a meaningful realization on any given platform. That means, developers have to take into account the properties of the infrastructure they are using, not only for the final implementation, but also already at a conceptual level during requirement analysis.

A conceptual model capturing the properties of a certain class of mobile computing platforms would be very helpful to the application development and the development of new mobile platforms. It would allow an understanding of the basic mechanisms and their suitability for a certain task. With suitable refinement and evolution support, the conceptual model can be mapped into a concrete platform specific model.

In reality, it is difficult and expensive to test the mobility support of a certain platform, which requires devices supporting wireless communication and specific tools to check the coordination logic of involved hardware and software components. Simulating the mobile platform can provide a simple and cheaper way to test the mobility aspects of the platform. Through this means, the context aspects of the platform like locations, network connections can be simulated directly, thus a dynamic execution environment can be provided for the context-aware applications, which is also difficult to test in reality.
Figure 1: Modeling and simulation framework

In this paper, we introduce our approach for analysis, design and simulation of mobile systems with the tool support of Fujaba. The approach is developed based on UML-like meta models and graph transformation techniques to support sound methodological principals, formal analysis and refinement. The approach includes two main parts: modeling and simulation. The modeling part is introduced in Sect. 2. Simulation is introduced in Sect. 3. Sect. 4. give the related work and Sect. 5 concludes the paper with finished work and future work.

2. MODELING OF THE MOBILE SYSTEM

Conceptual and concrete level modeling (as shown in Fig. 1) are the key parts of our approach [4]. The conceptual modeling of styles of mobile systems [5] is proposed as a way of capturing the properties of a certain class of mobile computing platforms. The conceptual model consists of two parts: a static structural model given by UML class diagrams whose instances represent the valid system configurations, and a dynamic behavioral model given by transformation rules over these instances, specifying the operations of the style. Typed graph transformation systems [8] will provide the underlying formal model and operational semantics. Informally, a typed graph transformation system $C = (TG, C, R)$, where $TG$ is a type graph (visualized by the class diagram) defining the architectural elements, $C$ is a set of constraints restricting their possible compositions, and $R$ is a set of graph transformation rules (given by pairs of object diagrams).

Our structural model consists of meta models at different levels contained in different packages. This allows us to separate different concerns, like software architecture, distribution and roaming, while at the same time retaining an integrated representation where all elements of a concrete model are presented as vertices of the same graph, i.e., an instance of the overall meta model. Based on this uniform representation, the different sub-models can be related by associations between elements belonging to different sub-models.

Based on the integrated representation of the different views in a single meta model, we can define the rules governing movement and connectivity as graph transformation rules typed over the corresponding package(s). A graph transformation rule $r: L \Rightarrow R$ consists of a pair of $TG$-typed instance graphs $L, R$ such that the intersection $L \cap R$ is well-defined. The left-hand side $L$ represents the pre-conditions of the rule while the right-hand side $R$ describes the post-conditions. In Fig. 2, the $moveIn$ rule is shown as an example: according to its precondition, expressed by the pattern on the left-hand side, there should be a Node $n$ and an Area $a$ whose types NT and AT should be connected by a locateAt link. That means the node is of a type that is supported by the area, like a cell phone in a GSM cell. In this case, the rule can be applied with the result of creating a new locateAt link between the two instances. This is expressed in the post-condition of the rule shown on the right-hand side.

In [5], we have presented a basic style of mobile information system for nomadic network, which is focussed on the roaming and connectivity of mobile hosts, i.e., hosts can change location and possible connections may vary according to this location change. Naturally, architecture and behavior of applications depend on the connectivity and location of their host computers. Our three-layered meta model captures these relations in the three packages Architecture, Connectivity and Roaming to present different viewpoints of the systems. The basic operations of the style include moveIn, moveOut, register, deregister, connect, disconnect and handOver.

The concrete model is based on a specific platform, e.g. Wireless CORBA. Concrete modeling of mobile system uses the same modeling technologies as conceptual modeling of styles. Given specified models of the platform, a prototype can be generated directly for the reference of implementation using code generation functionality provided by graph transformation tools like Fujaba [1].

The relationship between these two different layer models is refinement, e.g. the mobility and other aspects modeled in the conceptual model need to be mapped into a concrete design. Besides this, we focus on the behavior consistency check of the two levels (will be discussed in Sect 3.3.), i.e. the consistency check of the rules applied to the models, but not of the structural elements. This is because the construction elements inside the conceptual model do not need to be present in the concrete model, that makes the consistency check of the construction element not much meaningful in our approach.

3. SIMULATION

The operational semantics of the typed graph transformation system allows us to execute the models thus analyzing the system through simulation. In this section, we will introduce two ways to use the simulation through Fujaba: for validating the model and as an oracle for test the actual implementation.

3.1 Simulation for Validation
In graph transformation systems, many verification problems can be formulated as reachability (or non-reachability) properties of a given configuration in the system. A reachability property holds for a given graph transformation system $G = (TG, C, R)$ and a start graph $G_0$ if an instance graph that contains a certain target pattern is reachable by applying available transformation rules. This means that a system can evolve from the start configuration to the desired target configuration by performing given operations. In this way we can check, for example, if a required reference application scenario is realizable on the middleware, thus validating functional completeness of the model.

The object-oriented CASE tool Fujaba [1] supports the specification of a system using UML class diagrams and story diagrams, a combination of activity diagrams and collaboration activity diagrams (as a notation for graph rewriting rules). Executable Java source code can be generated automatically. To observe the running system, a Dynamic Object Browsing system (Dobs) supports the execution of rules, visualizing the effect of the application on an object graph representing the state of the Java heap.

We introduce how to use Fujaba to validate our defined specification. As shown in Fig. 3, we use the Fujaba class diagram editor to specify our meta model at first. Graph transformation rules (or operations) are defined in the Fujaba story diagram editor then. After generating and compiling Java code for the complete specification, we can start Dobs to execute the models. We can create an initial object configuration typed over the defined class diagrams. The initial object configuration represents a possible configuration of the system, which is also a start graph $G_0$ as defined before. Following the defined sequence of operations that describe the application scenario, we can then execute the sequence of operations against the start graph. For example, we can test if the terminal-initiated handoff scenario [3] (defined by a sequence of operations) is reachable by starting from the initial configuration. Through this way, we can test if the pre-defined scenario is supported by our specification, thus validating the functionality completeness of our model.

### 3.2 Simulation for Testing

All software testing methods depend on the availability of an oracle, that is, some methods for checking whether the system under test has behaved correctly on a particular execution. Executable formal specifications can be used as test oracles to produce the results expected for a test case. By comparing the result of a call to the actual implementation with the result of a call to the simulation, the test oracle can be used to check the correct execution of an operation.

We can extend the specified concrete model to a test oracle [3]. Since the concrete model is platform independent concerning the independency of specific programming languages, hardware platforms and concrete implementation methods, it can be reused as a reference to test the correctness of implementations on different platforms. As a test driver, a standard reference application shall be required. To facilitate the interaction between the reference application with our model (resp., the code generated from it), we need to provide an Application Programming Interface (API) that is consistent to the API provided by a middleware implementation. Using the same test application as a test driver for the implementation and for the defined model, the developers can trace errors in the execution and check the pre- and post-conditions of operations.

### 3.3 Wrapper for API–Behavior Consistency

The correct refinement of abstract conceptual styles into a concrete style is important, and the verification process is usually complicated. In order to automatize the consistency check between the conceptual and concrete models, we develop a Wrapper (in Fig. 1) to define the refinement relationship between these two models. Both the conceptual and the concrete model provide application programming interfaces through the operations defined via the rules, which are named Concrete API and Conceptual API as shown in Figure 1. As an adapter between Concrete API and Conceptual API, the wrapper encapsulates and maps the operations implemented in Concrete API to the operations defined in Conceptual API. Providing type transformation and semantic match, the Wrapper forwards operation calls to Conceptual API to the operation calls to Concrete API. In this way, the application can use the more abstract interface while the concrete operations offered by the platform remain hidden. This abstraction allows us to port the application to a new concrete platform API by means of a new wrapper, without changing the application itself. The wrapper can be also used to test, e.g., by means of a reference application, if the operations provided by Concrete API and Conceptual API are semantically compatible, therefore verifying the concrete style or the actual platform against the requirements expressed in the conceptual style.

### 4. RELATED WORK

Several proposals have influenced our approach. The general idea of modeling classes of systems with common structural and behavioral characteristics by a combination of metamodeling and graph transformation is due to [6], where it has been applied to software architecture styles. As mentioned before, the architectural style offers only a "logical" view of change; it does not take into account the properties of the "physical" distribution topology of locations and communication links. In our model, we extend the architecture style by adding mobility aspects, with the focus on roaming and connectivity issues.
Some of the techniques proposed by the AGILE project presented in [2] are close to our approach of modeling. AGILE develops an architectural approach by extending existing specification languages and methods to support mobility: UML stereotypes are used to extend UML class, sequence and activity diagrams in order to describe how mobile objects can migrate from one host to another, and how they can be hosts to other mobile objects. Graph transformation systems are proposed as a means to give an operational semantics to these extensions.

Other extensions are based on architectural description languages, like the parallel program design language CommUnity using graph transformation to describe the dynamic reconfiguration; Klaim as a programming language with coordination mechanisms for mobile components, services and resources; The specification language CASL as a means for providing architectural specification and verification mechanisms.

While Klaim and CASL are more programming and verification oriented, the approaches based on UML and CommUnity are at a level of abstraction similar to ours, but the goals are different: Our focus is to model a style of mobile applications, e.g., corresponding to a certain mobility platform, while the focus in the cited approaches is on the modeling of applications within a style more or less determined by the formalisms. Indeed, being based on a meta-model, our approach can easily specify styles exhibiting all kinds of features like QoS (as demonstrated in [5]) or more sophisticated aspects of context awareness, handOver operations within one or between different networks, etc.

Finally, our three-layered modeling approach provides a clear separation of the different views of software architecture, connectivity, and mobility, which is required in order to specify a physical phenomenon, like the loss of a signal, in relation with the intended reaction of an application or middleware platform, like the transfer of ongoing sessions to a new connection.

The idea of analysis and design of a system using a refinement approach is not new. Generally, the people focus on architecture refinement based on component and connectors where the construction elements in different abstract layers have direct corresponding relationship. In our framework, the construction elements inside the conceptual level do not need to appear in concrete level. We focus on the behavioral consistency check between conceptual and concrete level; this makes it easier to implement automatic consistency checks via testing.

5. CONCLUSION AND FUTURE WORK
The paper presents an approach for analysis, modeling and validation of mobile information systems with the tool support of Fujaba. Presenting the basic structures and operations common to a certain class of mobile computing platforms, with the refinement between conceptual and concrete models, the approach could support application development and the development of new mobile platforms. The approach also provides automatic analyze, validation and behavior consistency check with the support of Fujaba.

We have finished a conceptual model of a style for mobile system, which is a very important part of our approach. Not tailored towards a particular platform, the model reflects the properties of nomadic network, where mobile devices are supported by a fixed infrastructure. The validity of the model has been checked through the Fujaba simulation environment using an application scenario, where the explicitly modeled mobility and context aspects like locations, network connections can be simulated. Some parts of this work are presented in [5]. A concrete model of a specific platform (Wireless CORBA) has been also developed; the models are validated using Fujaba too [3].

A major issue of our future work is the improvement of the simulation to provide a more automated environment for the activation of methods. For example, it would be helpful to derive a sequence of operations from a sequence diagram execute it automatically based on a given the initial configuration. An improved, domain-specific visualization of object configurations is another aim, since the generic object-oriented representation is not concise enough for larger examples. The Wrapper for refinement behavioral consistency check between the conceptual and concrete layers need to be developed, which can be made as Fujaba plugin [1].

6. REFERENCES
[1] From UML to Java and Back Again: The Fujaba homepage. www.upb.de/cs/isileit