CONCEPTUAL MODELING OF STYLES FOR MOBILE SYSTEMS

A layered approach based on graph transformation

Reiko Heckel[1] and Ping Guo[2]
[1]*University of Dortmund, Germany(on leave from University of Paderborn), reiko@upb.de;*
[2]*University of Paderborn, Germany, Intl. Gradrate School of Dynamic Intelligent Systems, ping@upb.de*

Abstract: When designing a mobile application, we have to be aware of the properties and facilities of the target platform. At a conceptual level, this platform can be specified by a style, defining the structures and operations available to applications. In this paper, we use an UML-like meta model for the structural aspect and graph transformation rules over its instances to specify the dynamics of a style of mobile systems. The model is layered to separate clearly the software from the hardware and the geographic view of the system.

Keywords: mobility; QOS; meta modeling; graph transformation.

1. INTRODUCTION

Already today, the majority of computing devices is mobile. They include, besides smart cards with limited capabilities, laptops, handheld computers, and mobile phones, all equipped with communication and computation power beyond that of stationary computers a few years ago. In order to manage the resulting logic complexity of applications, conceptual modeling techniques are required like for the development of "stationary" software.

In addition, mobility creates new and diverse concerns. As stated in [19], mobility is a "total meltdown" of the stability assumed by distributed systems, the main differences being caused by the possibility of roaming and wireless connection [21]. Roaming implies that, since devices can move to

different locations, their context (network access, services, permissions, etc.) may change, and that mobile hosts are resource limited, for example, in computation power, memory capacity, and electrical power. Wireless connections are generally less reliable, more expensive, and provide smaller bandwidth, and they come in a variety of different technologies and protocols. All this results in a very dynamic software architecture, where configurations and interactions have to be adapted to the changing context and relative location of applications.

In order to provide continuous service to mobile devices, or compensate for the lack of it, a number of different solutions have been developed. Telecom systems like GSM, GPRS, or UMTS use handover protocols to provide continuous connectivity. Mobile IP extends seamless IP connectivity to mobile hosts. Besides, there exist middleware platforms supporting mobility at the application-programming level like J2ME, a reduced version of J2EE for resource-limited devices with support for wireless communication, and Wireless CORBA which supports terminal mobility through CORBA bridges and handover protocols.

However, even with dedicated middleware it is not possible in general to completely hide the consequences of mobility from the application which has to be aware of, and able to react to, changes in its context given by its current location, quality, cost, and type of available connections, etc. What is more, 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 requirements analysis.

In this paper, the conceptual modeling of *styles of mobile systems* is proposed as a way of capturing the properties of a certain class of mobile computing platforms. Such conceptual models consist two parts: a structural model given by UML class diagrams whose instances represent the valid system configurations, and a dynamic model given by transformation rules over these instances, specifying the operations of the style. Graph transformation systems [20] shall provide the underlying formal model and operational semantics.

Graphs provide a popular model for a variety of structures, including network topologies and software configurations. Their evolution can conveniently be specified by graph transformation rules that manipulate the graphs by means of pattern matching and rewriting. Such specifications have a formal operational semantics, can thus be executed by tools and analyzed for certain properties.

In order to employ graphs and graph transformations for our purposes, the different interpretations of their vertices as components, devices, areas, etc. and their edges as connectors, network connections, neighborhood relations, etc. have to be defined. This is done by a meta model consisting of three interconnected layers, structured by means of packages.

Rules formulated over this meta model shall describe a style of mobile systems given by a set of basic actions that can be invoked or observed by the application. This is comparable to the specification of an architectural style by graph transformation rules [14, 13, 6], but for the fact that the scope is not limited to the software architecture. Then, a mobile application scenario, expressed as a sequence of such basic actions, can be analyzed for its realizable in the style by constructing an execution of the corresponding rule sequence, either by means of a model checker or interactively with the help of a graph transformation tool. This aspect, however, is beyond the scope of this paper.

The paper is organized as follows. In Section 2, we introduce a basic style of mobile systems following the three-layered structure discussed above. In Section 3, this basic model is extended by taking into account quality-of-service information. An example is presented to illustrate this aspect. Section 4 discusses related work and Section 5 concludes the paper.

2. A BASIC STYLE OF MOBILE SYSTEMS

We give a conceptual model that captures the basic structures and operations typical to nomadic networks. Our conceptual model consists of two parts, a structural meta model represented by UML class diagrams and a dynamic model given by graph transformation rules.

2.1 Meta Model

Following [11], we use a meta model structured into three 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.

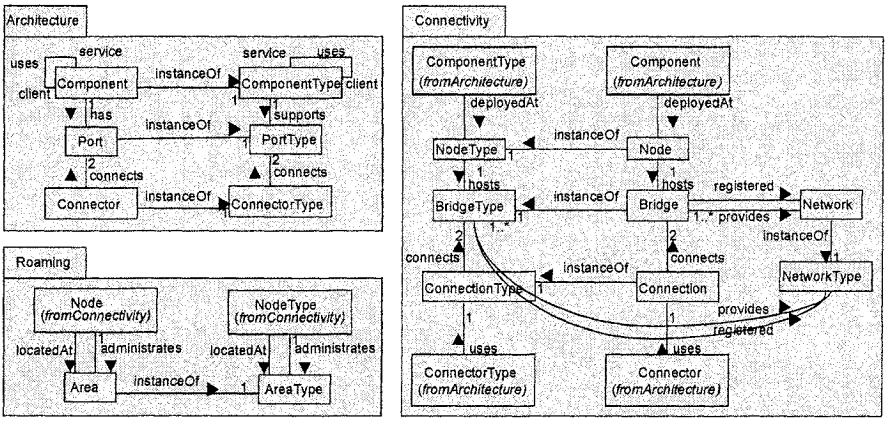

Figure 1. Architecture, Connectivity and *Roaming* package

Our basic style of mobile information systems is focused on the roaming and connectivity of mobile hosts, i.e., hosts can change location and possible connections may vary according to their relative location to each other. 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 Architecture package in Fig. 1 defines the architectural view, containing both a definition of an architectural model (meta classes *ComponentType*, *ConnectorType*, *PortType*) as well as of an individual configuration (meta classes *Component*, *Connector*, and *Port*), related by the meta association *instanceOf*.

The Connectivity package in Fig. 1 presents the distributed view of the system in terms of the concepts *Node*, *Bridge*, and *Connection*, paired as above with corresponding type-level concepts. A node is a (real or virtual) machine, accessible through bridges via connections. The typing means that we can distinguish, for example, between Ethernet, WLAN, or GSM-based connections, or between different kinds of machines like PCs, laptops, cell phones, etc. The package uses the elements *Component*, *ComponentType* from the Architecture package to be able to specify deployment using the *deployedAt* association. Bridges on client nodes need to be registered with a network in order to build up connections. Moreover, they are typed (e.g., an Ethernet bridge cannot connect to a GSM network). Bridges on server nodes provide network access to registered clients. A Connection is a physical network link that delivers communication services to Connectors at the software level.

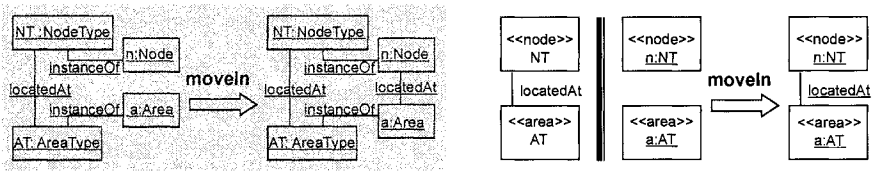

Figure 2. Transformation rule *moveIn*: meta model-based presentation (left) and UML-like concrete syntax (right)

The Roaming package in Fig. 1 defines the location and mobility of Nodes in terms of *Areas*, i.e., places where Nodes can be located. An area is defined by an administrative domain, like a cell managed by a GSM base station, or a Wireless LAN domain. Thus, there are different types of areas that may be overlapping. An area can have an administrative Node who serves the connections in this Area. This node does not need to be located inside the same area. We do not separate mobile from stationary hosts at this level. A node is mobile if it changes its location to a different area, a fact that is part of the dynamics of the model. This allows the added flexibility of considering, e.g., a laptop that does not move as a stationary device.

2.2 Rules

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). The basic operations of the style include *moveIn*, *moveOut*, *register*, *deRegister*, *connect*, *disconnect* and *handOver*. Due to space limitations, operations *moveOut*, *register*, *deRegister* and *disconnect* are omitted. *moveOut* and *disconnect* are dual to *moveIn* and *connect*, respectively. The *register* operation is responsible for subscribing a bridge on a client node with a network, so that the client gets permission to connect via this bridge. Operation *deregister* is, again, the dual.

In the left of Fig. 2 the *moveIn* rule is shown. According to its pre-condition, expressed by the pattern on the left-hand side, there should be a Node n and an Area whose types NT and AT should be connected by a *locatedAt* 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 *locatedAt* link between the two instances. This is expressed in the post-condition of the rule shown on the right-hand side. The dual operation *moveOut*, specifying the deletion of a *locatedAt* link, is omitted here.

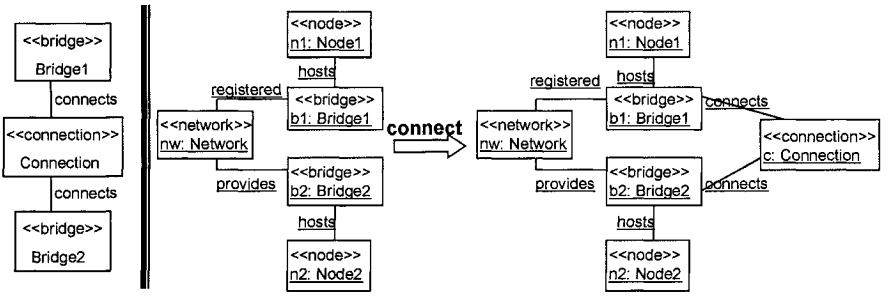

Figure 3. Transformation rule *connect*

In order to simplify the notation, we use a UML-like concrete syntax with separation of type and instance level and stereotypes to denote meta types. The corresponding presentation of the *moveIn* rule is shown in Fig. 2 on the right. The types declaration is separated from the actual rule by the black vertical line. The instanceOf relation is expressed in the standard way as *instance* : *Type*, while stereotypes like <<node>> represent meta types Node and NodeType.

Connecting is the act of building a network connection between two nodes through their bridges. The precondition of the rule in Fig. 3 requires the existence of a corresponding connection type between the two bridge types. Moreover, bridge b1 has to be registered with network nw provided by bridge b2. As postcondition, the desired connection c:Connection and its links are created. The dual rule *disconnect* for deleting a connection is, again, omitted.

These rules, as well as the following one, are typical for nomadic networks where a fixed infrastructure provides wireless connectivity to the mobile devices. Examples of such systems include mobile phone networks (GSM, GPRS, UMTS) or Wireless LAN.

Handover procedures are unavoidable in mobile system in order to realize seamless connectivity across different access points. At the physical link level, this means to preserve the connection between the moving terminal and its infrastructure [2], providing the same type of connectivity or switching from one type of connectivity to another as infrastructure coverage changes [4]. In higher-level protocols, like Wireless CORBA, one needs to support continuous object connectivity, using handover and tunneling between different bridges on the physical link layer. We give a handover rule typical for nomadic networks, where a fixed infrastructure provides connectivity to mobile hosts.

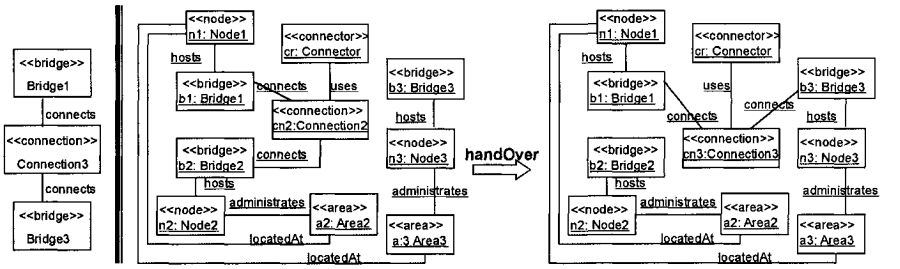

Figure 4. Transformation rule *handOver*

The rule shown in Fig. 4 explains how to maintain connectivity between administrative domains. It requires that node n1 is located in two areas served by two administrative nodes n2 and n3. Connector cr uses connection cn2 between bridge b1 and b2. The connection is replaced by cn3 of type Connection3 that, according to the types declared on the left, is permitted between bridges of type Bridge1 and Bridge3. The uses relation of the connector is transferred to the new connection.

3. TOWARDS A QOS-AWARE STYLE

In this section, we indicate a possible extension of our model towards a style of systems that are aware of quality-of-service (QoS) properties, requirements, and their matching. Different applications may have different QoS requirements, and the degree to which this aspect is transparent varies greatly in different systems. For example, [17] discusses the pros and cons of application-transparent vs. application-aware adaption to changes of QoS properties. Therefore, the handling of QoS is an important aspect to be modeled as part of a style of systems.

Without aiming at a complete model of this aspect, we want to demonstrate how the behavior of the operations of our style can be controlled by QoS properties. We present a basic model of QoS in mobile systems, again consisting of a static meta model and graph transformation rules. Then we illustrate the concepts by an example.

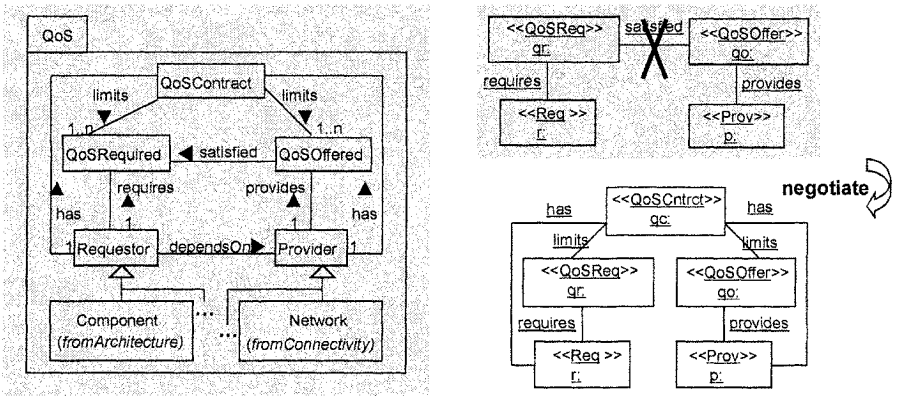

Figure 5. QOS package and transformation rule *negotiate*

3.1 Meta Model

The QoS package in left of Fig. 5 defines the basic concepts of QoS, i.e., requirements *QoSRequired* of service Requestors, properties *QoSOffered* offered by Providers, a dependency relation between Provider and Requestor, and a satisfaction relation between offered and required properties. The roles of Provider and Requestor can be played by any entity of the model, e.g., Network, Component, Connection, etc. A *QoSContract* is the result of a negotiation between Requester and Provider in case the offer does not meet the requirements, putting a lower limit to the first and an upper limit to the latter.

This quite general structure is a simplified version of the meta model of the UML profile for QoS [18]. An important difference is that, in our case, QoS is dealt with at the instance level, i.e., at run-time. For example, in [10] the authors stress that the major impact of mobility on QoS is due to the dynamic changes in the context of applications, and platforms like [17] monitor QoS changes at run-time to inform or adapt their applications. This does not exclude the specification and matching of QoS properties at the type level, but the instance-level is mandatory.

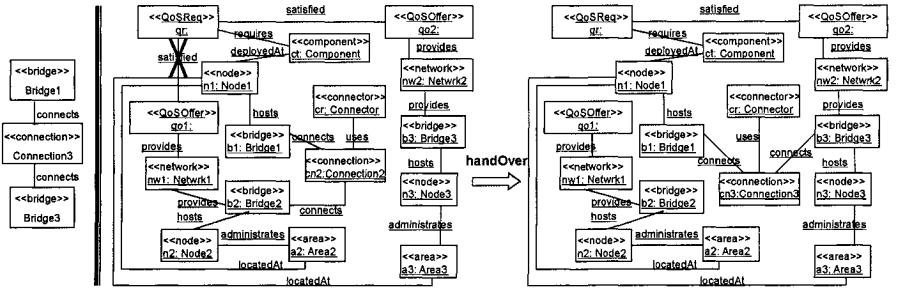

Figure 6. handOver rule with QOS

3.2 Rules

The effect of a negotiation between the QoS requester and provider when the offered QoS does not satisfy the requested QoS is defined by the rule *negotiate*. Since we consider QoS only at the instance level, the corresponding objects are labeled like *qo*: without type, but with stereotype <<QoSOffer>> referring to the meta type. The pre-condition of the rule in Fig. 5 on the right contains a negative application condition, represented by the crossed-out link, which makes sure that the rule is only applicable if the QoS requirements are not satisfied.

Note that match making in our model is abstracted through the *satisfies* association, while the detailed conditions for satisfaction and the protocols for negotiation are left open for possible refinements of the model towards a concrete platform.

As an example of how QoS may affect the behavior of operations of our style consider the extended *handOver* rule in Fig. 6. The idea is that the selection of the new bridge to connect to may depend on the QoS requirements of the application component as well as on the actual properties of available bridges. Both can, moreover, change over time.

Then, if the current network does not satisfy the current requirements, the responsible bridge will hand over to a bridge of a network where the requirements are satisfied. According to the left-hand side of the rule, it can only be applied if the quality of the current network is not satisfactory.

3.3 Example

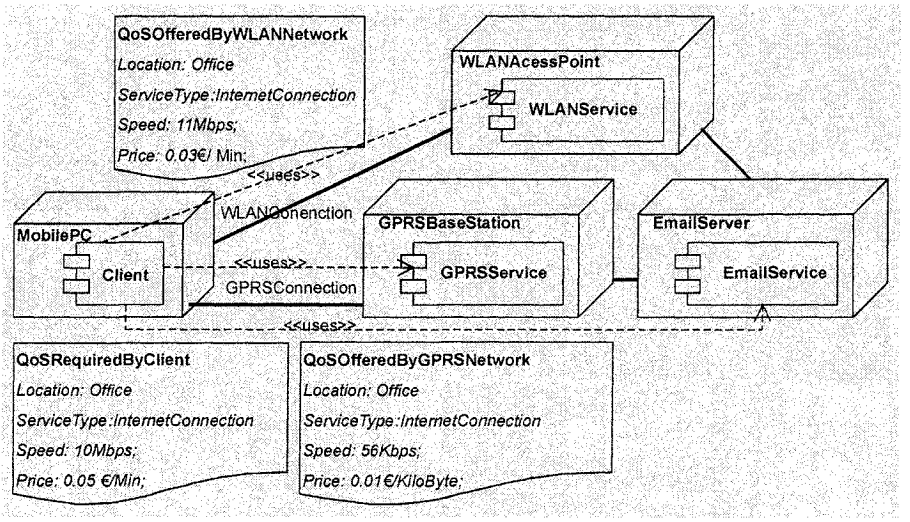

Figure 7. Application scenario with QOS (deployment diagram)

Let us conclude this section by giving an example for how to use our conceptual model in a concrete application. Assume a mobile PC equipped with Wireless LAN and GPRS cards, hosting a component that needs to access an Email server on a stationary host. We suppose that we have two areas for movement: the office and outside. Wireless LAN and GPRS networks are both available in the office, while outside only GPRS is available. Because of the higher speed and cheaper price, the system should use the Wireless LAN whenever available. However, when the user moves from the outside area to the office, the Email connector based on GPRS should not be interrupted while the underlying connection changes to Wireless LAN. The different QoS requirements and offers of the component and the two available networks are shown in the deployment diagram of Fig. 7.

This situation can be represented as an instance of the meta models introduced in this and the previous section. It should be clear that, being in the office, rule handOver in Fig. 6 will allow us to connect to the Wireless LAN, but not to the GPRS network. Fig. 8 shows a part of the configuration of the scenario before and after the application of this rule.

In the upper part of Fig. 8, the mobilePC has just moved from outside into the office. The lower part represents the configuration where the

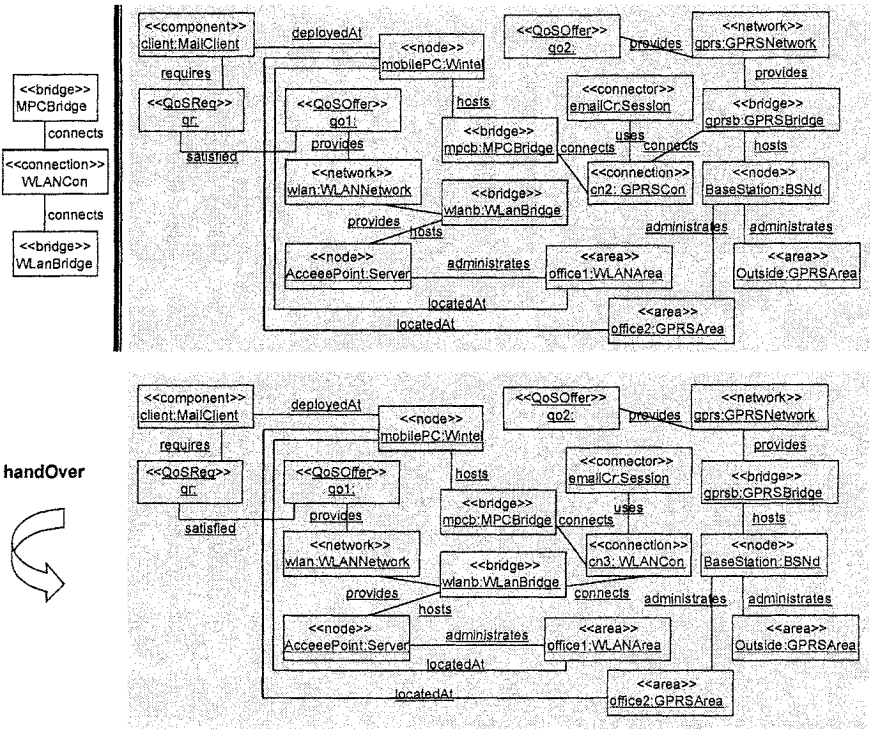

Figure 8. Configuration of the scenario before and after handover

mobilePC has been connected to the Wireless LAN network, while the Email connection has been be reserved.

4. RELATED WORK

The general idea of modeling classes of systems with common structural and behavioral characteristics by a combination of meta modeling and graph transformation is due to [14], where it has been applied to software architecture styles. For distributed systems, [13] argues that rules should be strictly local, covering only the operation of a single component, while global effects should be achieved by synchronization. This restriction, however, does not apply to the high-level specification of operations at the system level.

A number of different process calculi have been proposed to describe the interaction and movement of mobile processes, mostly by extending the π-

calculus [16, 12, 9]. In order to express runtime properties of mobile systems, some of these calculi have been complemented by logics [7, 15].

Apart from the fundamentally different appearance and style which, in our opinion, makes them harder to grasp for the average software engineer than our meta model-based approach, these process calculi with their associated logics have a complementary focus: They provide means for programming mobility in terms of the processes driving individual components or devices, rather than for its high-level conceptual modeling in terms of global pre- and post-conditions. In this sense, our model defines requirements, e.g., for handOver, while the actual protocol implementing the operation is more easily specified (and verified) in a process calculus.

Closer to our approach are some of the techniques proposed by the AGILE project [5], extending existing specification languages and methods. UML class, sequence, and activity diagrams are extended be features to describe how mobile objects can migrate from one host to another, and how they can be hosts to other mobile objects. In particular, their stereotypes <<mobile>> or <<location>> are not unlike our <<node>> and <<area>>. 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 [3, 5]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, again, more programming and verification-oriented, the approaches based on UML and CommUnity are at a level of abstraction similar to ours. However, 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 above) or more sophisticated aspects of context awareness, handOver operations within one or between different networks, etc.

Finally, our three-layered 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.

5. CONCLUSION

In this paper, we have presented a basic example for a style of mobile systems to illustrate how such styles can be defined using meta models and graph transformation. Not tailored towards a particular platform, the model reflects the properties of so-called nomadic networks, where mobile devices are supported by a fixed infrastructure. An extension towards QoS-dependent behavior of such systems is sketched and illustrated by an example.

In the future, we intend to use conceptual models like the one presented here as a means for classification, comparison, and improvement of mobility platforms. Moreover, the formal background of graph transformation systems can be exploited to analyze properties of systems by simulation.

Based on the operational semantics of graph transformation, a scenario represented by a trace of operations can be validated by executing the model starting from an initial configuration. Support for the execution of graph transformation systems is available, e.g., in tools like Fujaba [1] or PROGRES [22]. A major requirement here is a good visualization of configurations, since the meta model-based representation is not concise enough for larger examples.

REFERENCES

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

[2] I. Akyildiz, J. Mcnair, J. H, H. Uzunalioglu, and W. Wang. Mobility management in next-generation wireless systems. Proceedings of the IEEE, 87:1347– 1384, 1999.

[3] A. Lopes, J. Fiadeiro, and M. Wermelinger. Architectural primitives for distribution and mobility. In Proc. 10th ACM SIGSOFT symposium on Foundations of software engineering (FSE 2002), pages 41 – 50, Charleston, South Carolina, USA, 2002. ACM SIGSOFT.

[4] G. Alsenmyr, J. Bergstrm, and M. Hagberg. Handover between WCDMA and GSM, 2003.

[5] L. Andrade, P. Baldan, and H. Baumeister. AGILE: Software architecture for mobility. In Recent Trends in Algebraic Development, 16th Intl. Workshop (WADT 2002), volume 2755 of LNCS, Frauenchiemsee, 2003. Springer-Verlag.

[6] L. Baresi, R. Heckel, S. Thöne, and D. Varró. Modeling and validation of service oriented architectures: Application vs. style. In P. Inverardi and J. Paakki, editors, Proc ESEC 2003: 9th European Software Engineering Conference, pages 68–77, Helsinki, Finland, September 2003. ACM Press.

[7] L. Caires and L. Cardelli. A spatial logic for concurrency (part I). Information and Computation, 186(2):194 – 235, November 2003.

[8] L. Cardelli and A. Gordon. Anytime, anywhere. modal logics for mobile ambients. In 27th ACM Symposium on Principles of Programming Languages, pages 365–377. ACM, 2000.

[9] L. Cardelli and A.D. Gordon. Mobile ambients. In Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98. Springer-Verlag, Berlin Germany, 1998.

[10] D. Chalmers and M. Sloman. A survey of quality of service in mobile computing environments. IEEE Online Communication Surveys, 1(2), 1999.

[11] R. Heckel and G. Engels. Relating functional requirements and software architecture: Separation and consistency of concerns. Journal of Software Maintenance and Evolution: Research and Practice, 14(5), 2002. Special issue on Separation of Concerns for Software Evolution, edited by Tom Mens and Michel Wermelinger.

[12] M. Hennessy and J. Riely. A typed language for distributed mobile processes. In Proc. ACM Principles of Prog. Lang. ACM, 1998.

[13] D. Hirsch and U. Montanari. Consistent transformations for software architecture styles of distributed systems. In G. Stefanescu, editor, Workshop on Distributed Systems, volume 28 of Electronic Notes in TCS, 1999.

[14] Le Métayer, D. Software architecture styles as graph grammars. In Proceedings of the Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering, volume 216 of ACM Software Engineering Notes, pages 15–23, New York, October 16–18 1996. ACM Press.

[15] S. Merz, M Wirsing, and J. Zappe. A spatio-temporal logic for the specification and refinement of mobile systems. In Mauro Pezzé, editor, Proc. Fundamental Approaches to Software Engineering, 6th International Conference (FASE 2003), volume 2621 of LNCS, pages 87–101. Springer-Verlag, 2003. [16] R. Milner, J. Parrow, and D.Walker. A calculus of mobile processes. Information and Computation, 100:1–77, 1992.

[17] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R.Walker. Agile application-aware adaptation for mobility. In Proc. of the 16th ACM Symposium on Operating Systems Principles, pages 276–287, 1997.

[18] OMG. UML profile for modeling quality of service and fault tolerance characteristics and mechanisms, 2002. http://www.omg.org/docs/realtime/03-08-06.pdf.

[19] G.-C. Roman, G. P. Picco, and A. L. Murphy. Software engineering for mobility: A roadmap. In A. Finkelstein, editor, Proc. ICSE 2000: The Future of Software Engineering, pages 241– 258. ACM Press, 2000.

[20] G. Rozenberg, editor. Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations. World Scientific, 1997.

[21] M. Satyanarayanan. Fundamental challenges in mobile computing. In Symposium on Principles of Distributed Computing, pages 1–7, 1996.

[22] Andy Schürr, Andreas J. Winter, and Albert Zündorf. PROGRES: Language and environment. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools, pages 487–550. World Scientific, Singapore, 1997.