Modeling and Analyzing Mobile Software Architectures

Clemens Schäfer

Chair for Applied Telematics / e-Business*
University of Leipzig, Germany
schaefer@ebus.informatik.uni-leipzig.de

Abstract. The emerging behavior of a mobile system is determined by its software architecture (structure, dynamics, deployment), the underlying communication networks (topology, properties like bandwidth etc.) and interactions undertaken by the users of the system. In order to assess whether a mobile system fulfills its non-functional requirements like response times or availability already at design time, the emergent behavior of such a system can be simulated by using an architectural model of the system and applying an simulation approach where a network model and a user interaction model are used for providing the contextual information.
In this paper we show how such an architectural model can expressed in our ADL Con Moto, how functional and non-functional properties of an architecture can be modeled and how simulation of the mobile system can be used to yield the desired properties.

1 Motivation

Modeling the architecture of mobile distributed systems using a domain-specific architecture description language (ADL) is considered as an useful approach [3], since the influence of mobility emphasizes the necessity to examine functional properties of software architectures as well as non-functional properties. This corresponds to the fact that "mobility represents a total meltdown of all stability assumptions ... associated with distributed computing" [15], which subsumes the problems software engineers have to face in practice when they build mobile distributed systems. Examples for these problems are network structures, which are no longer fixed and where nodes may come and go, communication failures due to lost links over wireless networks, or restricted connectivity due to low bandwidth of mobile communications links. These all have in common that they affect the emergent non-functional properties of a system like performance, robustness, security or quality of service. Besides non-functional properties, these intrinsic challenges of mobile systems may also affect the functional aspects of a system, since a mobile system may have to provide extra functionality like replication facilities or caching mechanisms in order to ensure usability in situations

* The chair for Applied Telematics / e-Business is endowed by Deutsche Telekom AG.

where the aforementioned problems occur. With our ADL *Con Moto* (Italian for "with motion") we propose a language which enables system developers to address these issues during the early stages of system development in order to allow them to make appropriate design choices for the mobile system.

2 Introduction

Mobile systems show complex emergent behavior due to the combination of software aspects with telecommunication issues and the therefore eroding stability assumptions. In order to determine whether a mobile system fulfills non-functional requirements like response time or availability of service, a quite complex model of the system is needed.

1. The model must reflect the system's physical structure, comprising physical components (devices) and physical connectors (communication links, network topology) as well as the properties of these items like bandwidth or bandwidth distribution and computational resources, since for example a mobile component might take more time being executed on a mobile client compared to the execution on a server.
2. The logical structure of the system must be modeled in detail, comprising information about software components, their dependencies and deployment on the physical components and the possible changes in the deployment structure.
3. The model has to reflect the dynamics of the system, i.e. the behavior of the logical components, their interactions and the exchanged information.
4. Finally, user interaction with the system must be expressed, specifying how many users are existing and how these users interact with the system.

These aspects show that the challenge in modeling mobile system lies in the need to find an appropriate level of abstraction, since over-simplification will cause meaningless analysis results; however, too detailed models are not practical during the design process. Any modeling approach should remain as abstract and as free from technological implementations of real mobile systems as possible; nevertheless, realistic assumptions about the technological implementation of a mobile system are sometimes necessary to yield feasible simulation results.

The remainder of this paper is structured as follows. First, an overview about related work is given. Next, our approach for modeling mobile systems using Con Moto is presented. After depicting an example system and simulation results for this system, results are discussed.

3 Related Work

ADLs in general have been a topic of research in previous years. The necessity for modeling non-functional properties in architecture description has been recognized by Shaw and Garlan [16]. The classification work of Medvidovic and

Taylor [8] presents a sound compilation of properties of existing ADLs. From their work it becomes obvious, that none of the ADLs presented there is suitable for modeling dynamic aspects of mobile systems. In the past, this fact lead to the development of mobile ADLs which have recently been presented. The ArchWare project with its π-ADL [12] is one result of these efforts. Another mobile ADL can be found in the works of Issarny et al. [5]. Both present an ADL for mobile systems based on Milner's π-calculus [9]. These two ADLs have in common that they are able to model the dynamics of mobile systems, which is due to their theoretical foundation in the π-calculus. Although they vary in terms of elaboration and tool support, the fundamental difference—from the perspective of this paper—is the treatment of non-functional properties, which is absent in the π-calculus ADL approach. Issarny et al. address non-functional properties in their work, but the treatment of non-functional properties is bound to a global conformance condition, which must hold for a predefined set of non-functional properties assigned to components and connectors, and does not allow the composition of non-functional properties, which is novel in our approach. Besides the design of mobile ADLs there is other research in the area of non-functional properties of software systems. This work is mainly based on the Lamport's TLA+ language [6], which is a logic for specifying and reasoning about concurrent and reactive systems. Zschaler [17] presents a specification of timeliness properties of component based systems, but these as well as the underlying work of Aagedal [1], where the integration of TLA+ approach into architectural description is proposed, are not regarded further in our context, since the models in TLA+ lack the support for mobility. Other approaches based on Markov Chains and process algebras (e.g. the work of Hermanns and Katoen [4]) are not promising for out purposes, since fall short for the support for mobility.

4 Approach

In the following we describe the constituents of the Con Moto approach. All elements in the following are necessary to derive properties like bandwidth utilization, network congestion, dynamic evolvement of software deployment, transaction times or service availability for a system under analysis. Retrieving these properties during simulation is quite straight-forward if an appropriate representation of the mobile system and its usage is chosen.

Figure 1 shows an overview about the different elements of a Con Moto model and the simulation environment: The core architectural model is made from a behavioral and a structural specification of the system. This is due to the fact that in addition to the obviously existing structural model of mobile systems their behavior influences evolvment of the architecture and thus has to be modeled as well. Together with instantiation information, the simulator can create instances of the architectural model for simulation purposes. During simulation, communication network structures will be provided for the system as they are modeled in the network model. By applying user interactions by instantiating the Usage Patterns, the modeled system can evolve in the simulator

and the evaluation results can be calculated. In the following, we will present the different aspects of this model and exemplify their use by showing an example.

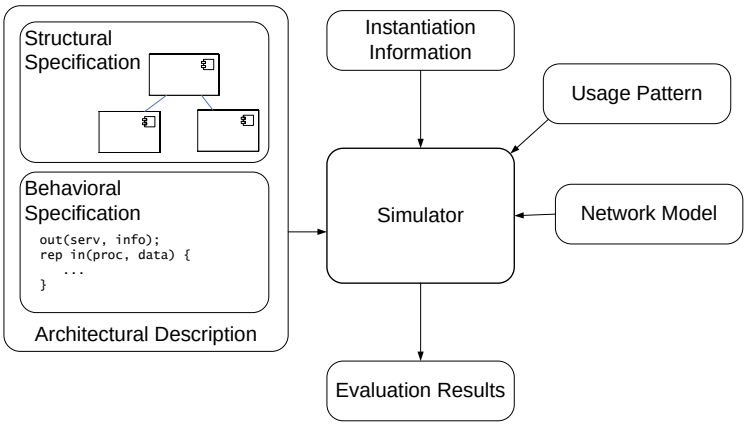

Fig. 1. Con Moto constituents

4.1 Behavioral Model

Mobile systems have to react to external conditions; the dynamically changing configuration is inherent to mobile systems. Therefore it makes sense to base architectural modeling on a behavioral model, assuming that any structural aspects like components or connectors can be seen as constraints for the behavioral model of the system.

Like other ADLs for mobile systems [13], we build our behavioral model on π-calculus. π-Calculus [10] is a process algebra with explicit support for mobility. It is based on communication primitives which allow the exchange of processes or communication nodes among processes. However, π-calculus in its full beauty offers features which are not necessary for our approach. Since we build a simulation environment, only constructs which reflect typical programming situations are used; others are discarded for the sake of simplicity. Such a restriction has also been done in the work of Pierce and Turner: with Pict [14] they present a π-calculus-based programming language, where they also omit some features of core-π-calculus, slightly reducing expressive power, but removing nondeterminism and making it appropriate for programmers.

As shown in Table 1, Con Moto provides different constructs for modeling processes: The output action allows the communication of an object over a so-called *Pin* in Con Moto (in π-calculus, the pins are called *names*). Other than in Pict, we only allow the synchronous output like in π-calculus, since we decouple input and output by means of the connectors.

Similar to Pict, we restrict π-calculus's replication prefix to input statements. Hence we do not allow the replication of processes; nevertheless, new processes can be created together with input operations, which is a quite realistic assumption, as it allows easily the creation of processes which respond to input data. The choice operator as a source for nondeterminism is omitted, but a if/then/else construct is added.

π-Calculus	Con Moto	
$\overline{x}y$	`out(x,y)`	synchronous output
$x(y)$	`in(x,y)`	input
$e_1 \mid e_2$	`par e1, e2`	parallel composition
$(\nu x)e$	`new x; e`	channel creation
$!x(y).e$	`rep in(x,y) e`	replicated input

Table 1. Notation

Modeling behavior includes messages that will be exchanged by processes will be implemented in Con Moto. Usually, abstractions of real-world messages are used in such situations; only that portion of a message is modeled, which is absolutely necessary to reflect the message's impact on control flow and behavior of the system. In Con Moto, we also specify meta-information about the size of messages, because in simulation situation the real-world size of such objects is necessary for simulation, hence supporting non-functional properties, since these meta information can be used e.g. by the network part of the model to calculate transmission times etc..

4.2 Structural Model

Having identified the processes as basis for the model of a mobile system, structural information has to be added since a solely behavioral view of the system would be unappropriate. Therefore, a structural model of the mobile system is set up. The challenges are twofold: On the one hand we now need an abstraction which allows us to set up a decomposition of a mobile system and on the other hand we need some decision on what the smallest entity of mobile code is.

Structural aspects have been considered in all ADLs so far. It is commonly accepted that an structural model comprises components, connectors and configurations. The components are the *locus computandi*: calculations are preformed on the components, whereas connectors model the communication relationships among components. Configuration can be seen as the state of a system and represents all interconnections between components by means of connectors.

Components For modeling mobile systems we have to clarify the notion of components and connectors. In Con Moto, we distinguish between *physical components* and *logical components*. Physical components are devices like PDAs or

servers, are constrained in their resources (memory size, CPU power etc.) and act as execution environment for logical components. Logical components model software components. They do not have resource constraints in our understanding and can occur as components and component instances. Instances of logical components have a state. In order to allow communication, physical as well as logical components have ports, which are aggregations of ports and pins, which finally allow the interconnected processes to communicate.

Connectors In Con Moto there are two different kinds of connectors, namely *physical connectors* and *logical connectors*. Logical connectors are used for communication between logical components and are ideal: they have an unbounded bandwidth and null latency. In contrast, physical connectors connect physical components and these are not ideal, having a limited bandwidth and a latency time greater than zero.

Logical connectors can be embedded in physical connectors. This is necessary, if logical components on different physical components shall communicate. The logical connector between the two connected logical components is embedded in the physical connector between the two physical components, which act as the execution environments for the two logical components.

Mobility Components are the smallest entity of mobile code in Con Moto. We assume that the *component* should be the element which is mobile. We do not take the extreme view of Mascolo et al. that every line of code is potentially mobile [7], because we want to model systems where this assumption would be unrealistic. We allow logical components as well as logical component instances to be communicated among processes. The same is true for logical connectors. This allows us to cover all kinds of mobility which are shown in the work of Fuggetta et al. [2]:

- *Client-server*, where a data file f is transferred from a node n_u to a node n_p. A program p executes on node n_p and the results are transferred to node n_u. The client on node n_u controls the operation. This is the situation as shown in our example below.
- *Remote evaluation*, where a program p is transferred from node n_u to node n_p, and executed there. Results are returned to n_u. The client controls the operation. Using Con Moto, this can be expressed by sending a logical component (which is the program p) to the computing node.
- *Code-on-demand*. Data file f and program p are transferred to n_u and executes there. The user demanding the code controls the operation.
- *Mobile agents*. Program p is transferred to n_f and executes there. Results are transferred to n_u. The agent itself controls the operation.

Configuration It is obvious that configuration of mobile systems evolve over time, since components can connect and disconnect to other components due

to their behavior. For mobile systems, however, developers usually express constraints on the possible configurations which might occur. By means of deployment diagrams like in UML 2.0 [11], developers of systems can express where components are deployed, hence which logical components are placed on which physical components. However, to be able to express constraints for configuration evolvement, this is not sufficient. Besides expressing an initial state of the deployment, there should be the possibility of expressing where components may be deployed during runtime, because then and only then runtime checks are possible whether the configuration of an mobile system evolves correctly.

Architectural Connection Architectural connection, hence the way how components are connected to each other by means of connectors, is a crucial aspect for mobile systems, since here all imponderabilities of mobility arise. For realistic systems, there may be many and complex dependencies among logical components leading to many logical connections. Physical connections are fewer: usually only a small number of physical connectors from among physical components.

In our system, logical connections must be embedded in physical connections; logical connections hence cannot be ideal–there is no synchronicity or parallelism.

In order to allow different communication protocols like synchronous calls (e.g. Remote Procedure Calls, Service Invocation) and asynchronous communication (events), our approach using pins where processes can exchange information is sufficient. Nevertheless, when a system is modeled on a quite high-level basis, there is the requirement for provides– and uses–interfaces and for services.

In order to provide a general basis, we introduce in Con Moto the possibility of ports which can consist of other ports and pins. By expressing bind rules, high-level ports can be connected, and by resolving the port hierarchy and subsequent application of binding rules various pins will be connected.

5 Example System

For illustration purposes we will use a simple example system. This example system is a mobile client/server system. The users of the mobile system carry mobile devices, which are connected to a server via mobile communication links; in our example, we provide either an GPRS link, which has a rather low bandwidth, and an UMTS (3G) link, which has a higher bandwidth. There are three software components in the example system: a user interface component (UI) is deployed on the mobile devices; a database component (DATA) is deployed on the server. The actual business logic of our system is captured in the component BUSINESS, which is a mobile component and thus can be either deployed on the server or on the mobile devices. When the user invokes a service of the UI component, a request is sent to the BUSINESS component (either on the mobile device or on the server). This component itself invokes a service of the DATA component before it returns its calculation results to the UI component. The structure of the example system is shown in Figure 2.

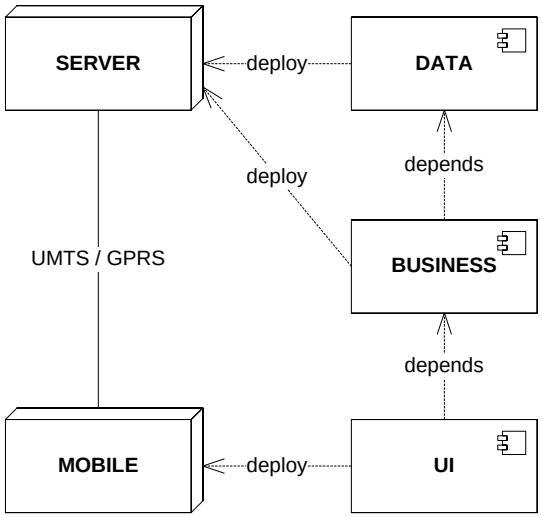

Fig. 2. Example system

5.1 Modeling in Con Moto

At the end of this paper, the Con Moto code, which is actually a document in an XML dialect, of the described example is shown. The two hardware components `MOBILE` and `SERVER` are declared in the section `<physical-components>`. For both, their CPU power is set and the possible connections to the network, which ends up in physical connectors during simulation. The `<network-access>` for `MOBILE` allows connection either to UMTS or GPRS network, the `SERVER` can only connect to the WAN.

The actual network model is given in the section `<network-config>`. Here, the network types UMTS, GPRS and WAN are defined. For all these network types, the bandwidth is specified (10.0, 2.0 and 1000.0 kBit/s). Latency times are not given. An additional network node named `backbone` is also given. All network connections via UMTS, GPRS and WAN automatically connect to this backbone, allowing to address any device from any other device which is connected to the network, i.e. physical components can communicate when they have connected to the network–which is a model similar to the internet. For UMTS and GPRS nodes in the network, we define that these nodes are equally distributed, which is necessary information if during simulation the number of network nodes is increased.

By introducing ports and port hierarchies in the section `<connection>` it is possible to have complex ports which act as an method provider interface or method invoker interface. By specifying macros for ports a certain behavior can be implemented in the port definition and easily reused in the actual process definition. In the example, the invokation of a service is modeled as a macro in

port `methodInvoker`. Since port `methodsProvider` has an extendable process which provides the counterpart for this macro, method invocation, waiting for execution and returning of a method result can be specified in π-calculus using `in` and `out` command on pins. In the processes in definition of the logical components, however, these macros and processes can be reused, yielding a code which is structurally equivalent to code in an imperative programming environment.

The logical components `DATA`, `BUSINESS` and `UI` are specified in the section `logical-components`. For the components `BUSINESS` and `UI` startup processes are defined, which execute when the components are deployed. During these processes, lookups of the components (`BUSINESS` in case of `UI` and `DATA` in case of `BUSINESS`) are performed and the logical connections to the components are established.

The processes of the `methodProvider` ports are extended for implementing services on components, such that the action which is to be undertaken after a service has been called is implemented in the processes on the logical components. On `DATA` the service `getData` sets a size of the return package of 100 bytes and blocks the CPU for 100ms. This return package size is used by the simulator to calculate the transmission time through the network. On `BUSINESS` the service `getInfo` makes a call to `getData` before a return package size of 5000 bytes is set and the CPU is blocked for 500ms.

5.2 Simulation

We have simulated the example system described here using our Con Moto simulator and have varied the users (and respectively, the `MOBILE` devices) from 10 to 150. The users use the system as modeled by a Poisson-process with an arrival rate of 10 per hour. The simulations have been performed for an time resolution of 1ms, and each simulation took not more than approximately 10 seconds on a 2 GHz Pentium PC, using a prototypical implementation of the simulator written in Java. Figure 3 shows the simulations results, meaning that starting with 90 users, the system gets increasingly congested and the response times of the services at the UI component increase drastically. Differences can be seen in the response times of GPRS and UMTS, which is due to the higher bandwidth of UMTS compared to GPRS.

6 Discussion

In this paper we have described how mobile systems can be modeled using the Con Moto approach with the goal of determining quality-of-service parameters during design time by means of an simulation approach. By basing an architectural description on π-calculus and making a clear distinction between logical and physical components and connectors, modeling of mobile systems on a quite high level is possible with feasible effort. First simulation results on an toy example system show that the general approach is promising. Nevertheless, further formalization of the approach is necessary and subject to ongoing work.

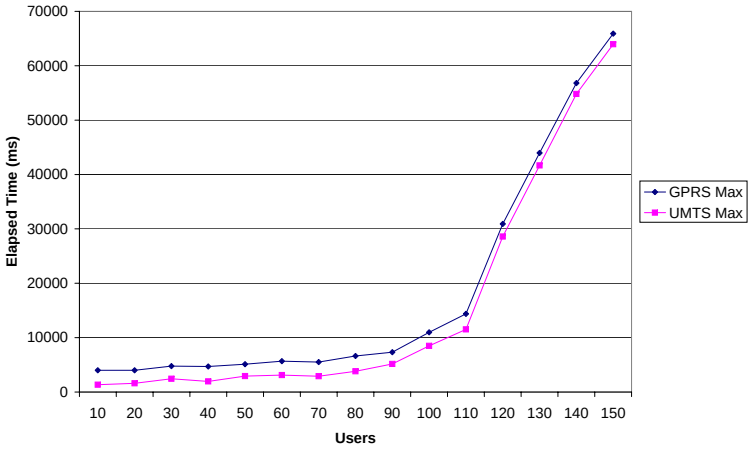

Fig. 3. Simulation result

Areas of further work are the discussion of models for physical communication channels. So far, we assume just constant bandwidth and latency time, but more complex models of modeling transmission characteristics of communication channels and–especially–availability characteristics of these channels are necessary for realistic simulation results. The area of user interaction with a mobile system is also part of further investigation, since not only the stochastic processes for user behavior need careful consideration–also the question how to derive user interaction patterns suitable from simulation from business process models is interesting. Finally, evaluation of the approach by comparing simulation results to real-world measurements is a future task.

References

1. J. Ø. Aagedal. *Quality of Service Support in Development of Distributed Systems.* PhD thesis, University of Oslo, 2001.
2. A. Fuggetta, G. P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.
3. V. Gruhn and C. Schäfer. Architecture Description for Mobile Distributed Systems. In *Proceedings of the Second European Workshop on Software Architecture (EWSA 2005)*, pages 239–246. Springer-Verlag Berlin Heidelberg, 2005.
4. H. Hermanns and J.-P. Katoen. Performance Evaluation := (Process Algebra + Model Checking) × Markov Chains. In *Proceedings of CONCUR 2001*, LNCS 2154, pages 59–81. Springer-Verlag Berlin Heidelberg, 2001.
5. V. Issarny, F. Tartanoglu, J. Liu, and F. Sailhan. Software Architecture for Mobile Distributed Computing. In *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, pages 201–210. IEEE, 2004.
6. L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley, 2002.

7. C. Mascolo, G. P. Picco, and G.-C. Roman. A fine-grained model for code mobility. In *ESEC/FSE-7: Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 39–56, London, UK, 1999. Springer-Verlag.
8. N. Medvidovic and R. N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, Januar 2000.
9. R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
10. R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
11. OMG. Unified Modeling Language (UML) Specification: Superstructure, Version 2.0 (formal/05-07-04).
12. F. Oquendo. π-ADL: An Architecture Description Language based on the Higher-Order Typed π-Calculus for Specifying Dynamic and Mobile Software Architectures. *ACM Software Engineering Notes*, 29, Mai 2004.
13. F. Oquendo. π-ADL: An Architecture Description Language based on the Higher-Order Typed π-Calculus for Specifying Dynamic and Mobile Software Architectures. *ACM Software Engineering Notes*, 29, May 2004.
14. B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.
15. G.-C. Roman, G. P. Picco, and A. L. Murphy. Software Engineering for Mobility: A Roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 241–258. ACM Press, 2000.
16. M. Shaw and D. Garlan. Formulations and Formalisms in Software Architecture. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 307–323. Springer, 1995.
17. S. Zschaler. Formal specification of non-functional properties of component-based software. In J.-M. Bruel, G. Georg, H. Hussmann, I. Ober, C. Pohl, J. Whittle, and S. Zschaler, editors, *Workshop on Models for Non-functional Aspects of Component-Based Software (NfC'04) at UML conference 2004*, September 2004.

7 Example Code

```
<system>
  <connection>
    <port-role name="in" />
    <port-role name="out" />

    <port-role name="methodsInvoker" extends-role="out" >
      <ports name="methodInvoker" />
    </port-role>

    <port-role name="methodInvoker" >
      <pin name="call" />
      <pin name="return" />

      <macro>
        <parameter name="argument" />
        <result name="result" />
        <pi>
          out(call, argument);
          in(return, result);
```

```
      </pi>
    </macro>
  </port-role>

  <port-role name="methodsProvider" extends-role="in" >
    <ports name="methodProvider" />
  </port-role>

  <port-role name="methodProvider" >
    <pin name="invoke" />
    <pin name="response" />

    <process>
      <pi>
        object arg, result;
        rep in(invoke, arg) {
          <extension-point />
          out(response, result);
        }
      </pi>
    </process>
  </port-role>

  <bind-rule>
    <scope>
      <from>methodsInvoker</from>
      <to>methodsProvider</to>
    </scope>
    <bind>
      <from>methodsInvoker.methodInvoker</from>
      <to>methodsProvider.methodProvider</to>
    </bind>
  </bind-rule>

  <bind-rule>
    <scope>
      <from>methodInvoker</from>
      <to>methodProvider</to>
    </scope>
    <bind>
      <from>methodInvoker.call</from>
      <to>methodProvider.invoke</to>
    </bind>
    <bind>
      <from>methodInvoker.response</from>
      <to>methodProvider.return</to>
    </bind>
  </bind-rule>
</connection>


<network-config>
  <passive-node name="backbone" />

  <active-node name="UMTS">
     <multiplicity>0.5</multiplicity>
     <auto-link>
       <node>backbone</node>
       <bandwidth>10.0</bandwidth>
     </auto-link>
  </active-node>

  <active-node name="GPRS">
     <multiplicity>0.5</multiplicity>
     <auto-link>
       <node>backbone</node>
       <bandwidth>2.0</bandwidth>
     </auto-link>
```

```
  </active-node>

  <active-node name="WAN">
     <multiplicity>unbounded</multiplicity>
     <auto-link>
       <node>backbone</node>
       <bandwidth>1000.0</bandwidth>
     </auto-link>
  </active-node>

</network-config>


<logical-components>
  <component name="DATA">

    <port type="methodProvider" name="getData">
      <extend-process>
        <pi>
          result.size = 100;
          useCpu(100);
        </pi>
      </extend-process>
    </port>
  </component>

  <component name="BUSINESS">
    <size>200</size>

    <start-process>
      <pi>
        PhysComp remoteHW = lookupPhysComp("SERVER");
        LogComp remoteSW = remoteHW.lookupLogComp("DATA");
        connect(this.getData, remoteSW.getData);
      </pi>
    </start-process>

    <port type="methodInvoker" name="getData" />

    <port type="methodProvider" name="getInfo" >
      <extend-process>
        <pi>
          object res;
          object par;
          res = getData(par);
          result.size = 5000;
          useCpu(500);
        </pi>
      </extend-process>
    </port>
  </component>

  <component name="UI">
    <port type="methodInvoker" name="getInfo" />

    <start-process>
      <pi>
        PhysComp remoteHW = lookupPhysComp("SERVER");
        LogComp remoteSW = remoteHW.lookupLogComp("BUSINESS");
        connect(this.getInfo, remoteSW.getInfo);
      </pi>
    </start-process>

    <pin name="action">
      <process>
        <pi>
          object dummy;
          rep in(action, dummy) { getInfo(dummy); }
```

```
          </pi>
        </process>
      </pin>
    </component>
  </logical-components>

  <physical-components>
    <component name="MOBILE">
      <memory>unbounded</memory>
      <cpu>10</cpu>

      <network-access>
        <xor>
          <type>UMTS</type>
          <type>GPRS</type>
        </xor>
      </network-access>

      <logical-component-deployment>
        <name>UI</name>
        <instance>on-start</instance>
      </logical-component-deployment>

      <logical-component-deployment>
        <name>BUSINESS</name>
        <instance>client-controlled</instance>
      </logical-component-deployment>

    </component>

    <component name="SERVER">
      <memory>unbounded</memory>
      <cpu>1000</cpu>

      <network-access>
         <type>WAN</type>
      </network-access>

      <logical-component-deployment>
        <name>BUSINESS</name>
        <instance>on-start</instance>
      </logical-component-deployment>

      <logical-component-deployment>
        <name>DATA</name>
        <instance>on-start</instance>
      </logical-component-deployment>
    </component>

  </physical-components>

</system>
```