Model-Driven Engineering of Ambient Intelligence Systems with SDL: Design, Implementation, and Performance Simulation

I. Fliege, A. Geraldy, R. Gotzhein, T. Kuhn, C. Webel

Technical Report 342/05

Model-Driven Engineering of Ambient Intelligence Systems with SDL: Design, Implementation, and Performance Simulation

I. Fliege, A. Geraldy, R. Gotzhein, T. Kuhn, C. Webel

Computer Science Department, University of Kaiserslautern, Kaiserslautern, Germany
{fliege,geraldy,gotzhein,kuhn,webel}@informatik.uni-kl.de

Technical Report 342/05

Computer Science Department
University of Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany

Technical Report

Model-Driven Engineering of Ambient Intelligence Systems with SDL: Design, Implementation, and Performance Simulation

I. Fliege A. Geraldy R. Gotzhein T. Kuhn C. Webel

{fliege, geraldy, gotzhein, kuhn, webel}@informatik.uni-kl.de

Abstract

OMG's Model-Driven Architecture™ initiative is a decisive step to place the abstract, formal system model in the center of the development activity. In this paper, we build on and refine this idea, using ITU-T's SDL as design language. We present a comprehensive, holistic model-driven approach that covers the entire development trajectory, from the initial requirements to the system in execution, with specific focus on the peculiarities of ambient intelligence systems (e.g., scarce and dynamic resource situations, adaptivity, cross-layer integration). With our reuse methodologies, we support transformations during the early development steps. With our tool chain, we replace manual coding steps entirely. Furthermore, we address model-driven performance simulation, and demonstrate that the platform-specific model can be used as a common code base for the generation of simulation and production code. All steps are illustrated by excerpts from the development of the Assisted Bicycle Trainer, an ambient intelligence system supporting the training of a group of cyclists.

1 Introduction

Model-driven development is a software engineering approach that places the abstract, formal system model in the center of the development activity [1]. The objective is that models guide and direct all development activities, ranging from system design over code generation and deployment to system maintenance, and resulting both in quality improvements and productivity increases. Model-driven development is promoted by the Object Management Group in the Model-Driven Architecture^{TM} initiative [8], and supported by the UML 2.0 [9]. Also, the model-driven approach is being applied in the telecommunications industry, where it is combined with SDL, the Specification and Design Language [5], internationally standardized by the International Telecommunications Union (ITU).

A key idea of model-driven development is the separation of the system design from platform-specific details. While this seems feasible for platforms offering sufficient flexibility and resources e.g., J2EE, .NET, CORBA [8], it is more difficult for restricted, hardware-specific and heterogeneous platforms as found, for instance, in the area of *Ambient Intelligence (AmI)* [7]. AmI systems provide ubiquitous services that enhance human capabilities and the quality of life by distributed intelligent electronic environments composed of unobtrusive electronic devices, embedded into the surrounding world and interconnected through wireless ad-hoc networks. They are sensitive to the presence of humans and objects, can react in response to a person and its environment, and can be tailored to a person's needs. AmI systems will have an impact on all spheres of our life, including professional work, leisure activities, public health, transportation, and communication.

Due to hardware-specific and heterogeneous platforms, model-driven development is a major challenge in the area of Ambient Intelligence, with specific focus on the following aspects:

- *Model-driven adaptation.* AmI systems operate under scarce and/or dynamically varying resources, such as computation power, memory, connectivity, bandwidth, and energy. System functionalities strongly depend on the current resource situation. Therefore, AmI systems have to adapt to the resource situation by upgrading or downgrading their quality of service. As a consequence, resources and adaptation mechanisms have to be incorporated into the system model, using adequate abstractions.

- *Model-driven cross-layer integration.* The nodes of AmI systems typically communicate via ad-hoc networks, i. e. wireless, mobile, self-organizing networks that come into existence by the mere presence of nodes with matching communication facilities. Again, resources are a decisive factor, requiring highly specialized communication protocols that can adapt to the resource situation. As a consequence, the communication system has to be incorporated into the system model. As a

further consequence, the abstraction layers of AmI systems should be harmonized, requiring cross-layer integration to be represented in the system model.

- *Model-driven performance simulation.* AmI systems usually operate under certain real-time constraints, which calls for performance assessments. These assessments have to take resource bottlenecks, such as communication bandwidth and connectivity, into account and should faithfully reflect the behavior of the system in operation. However, waiting until the system is actually deployed before the assessment becomes feasible may not be a good idea. Instead, the assessment could be based on a performance simulation of the system model. As a consequence, all functionalities that have a major influence on the system performance are to be incorporated into this model. To run performance simulations, this model is extended by abstract, sufficiently accurate resource models.

In this paper, we build on and refine the idea of placing the abstract, formal system model in the center of the development activity, using SDL as design language. Our process model takes the OMG MDA as a starting point, and refines it on the implementation side to meet the requirements of AmI systems. The transformation of the process independent model takes place in several steps, and is supported by heuristics and reuse methods as well as by a syntactically and semantically integrated tool chain. We have complemented existing commercial development tools, replacing extensive manual coding steps by generation techniques. Thus, the entire development trajectory from the system requirements to the system in execution is supported both from the methodological and the tooling side, yielding a comprehensive, holistic model-driven engineering approach for the development of AmI systems. The approach is illustrated by the development of the Assisted Bicycle Trainer, an AmI system supporting the training of a group of cyclists, from the initial requirements to the system in execution.

To assess the performance of AmI systems, we use the platform-specific design model as a starting point. This design model is sufficiently detailed for this purpose, and includes, in particular, adaptation mechanisms, which are essential for AmI system operating under scarce and dynamic resource situations, cross-layer integration aspects, and abstract resource models. We will show that it is possible to use the design model as a common code base for the generation of simulation and production code. Furthermore, the same compiler is used to generate this code. Both measures increase confidence that the results of the performance evaluation hold for the networked system in operation. To run performance simulations of SDL design models, we have extended the network simulator *ns-2* [13], which is widely used for the simulation of computer networks. Performance simulations of the Assisted Bicycle Trainer give evidence for the feasibility and usefulness of this approach.

The rest of this paper is organized as follows. In Section 2, the AmI system Assisted Bicycle Trainer (ABT), which has been engineered with our approach, is briefly introduced. In Section 3.1, we elaborate on model-driven design with SDL, and illustrate all steps by excerpts of the ABT specifcations. Model-driven implementation is addressed

in Section 3.2, featuring several new tools for the automatic generation of a prototype system. In addition, an approach for model-driven perfomance simulations is presented and applied in Section 3.3. The paper ends with a summary and an outlook (Section 4).

2 The Assisted Bicycle Trainer

To illustrate our model-driven engineering approach, we will show excerpts from the development of the Assisted Bicycle Trainer (ABT), an ambient intelligence system for the training of a group of cyclists [7]. In a typical training scenario, a group of up to 30 cyclists covers a distance of up to 200 km, with a varying road profile. For best training effects, each cyclist should ride with an individual average and maximum pulse rate. The pulse rate depends on various parameters, in particular on speed, head wind, road incline, and physical condition of the cyclist.

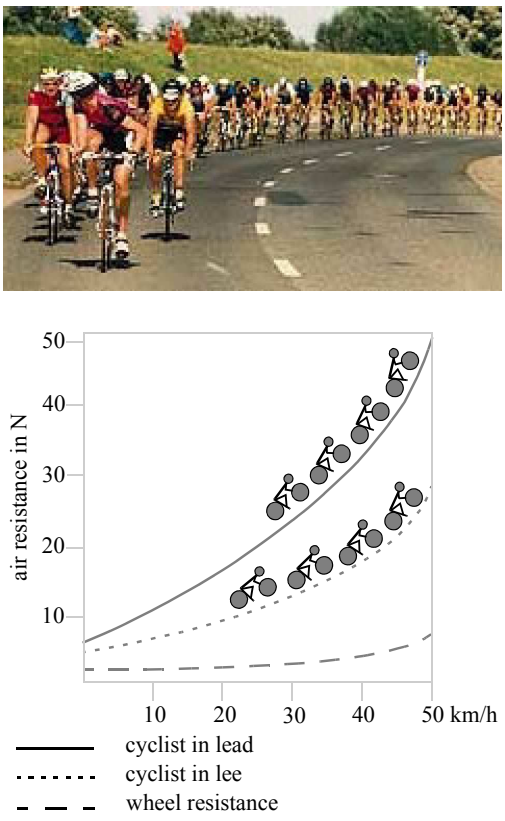

Figure 1: Assisted bicycle training

The objective of the ABT is to improve the training effect such that each cyclist is as close to his individual target mean pulse rate as possible, without exceeding his maximum pulse rate. To achieve this objective, the ABT dynamically collects status data of each cyclist, and displays a summary of these data to the human trainer accompanying the group of cyclists by car. Based on this information, the trainer may adjust training parameters, for instance, by ordering the group to change speed, or by ordering a particular cyclist to take the lead, exposing him to the headwind, while all others can exploit the slipstream and thus need less pedal power. Orders of the trainer are shown on small displays attached to each bicycle. The ABT is a self-organizing system, supporting, in particular, dynamic group formation and mobility. Communication among cyclists and human trainer is via wireless ad-hoc network.

3 Model-Driven Engineering with SDL

In this main section of the paper we present our model-driven engineering approach, elaborating on process steps, tools, and our case study.

3.1 Model-Driven Design

In this section, we survey the early phases of the model-driven development process model, which are fully in line with the OMG MDA (see Figure 2). The computation independent model (CIM) is expressed by use cases and informal text. Additionally, message sequence diagrams may be added. For the specification of the platform-independent and platform-specific models (PIM and PSM), we use SDL [5], ITU-T's Specification and Description Language for distributed systems and communication protocols. Over a period of more than 20 years, SDL has matured from a simple notation to describe asynchronously communicating finite state machines to a sophisticated formal specification language, with graphical syntax, data types, structuring concepts, support for reuse, and commercial tool environments. Major application areas are telecommunication and automotive systems [10].

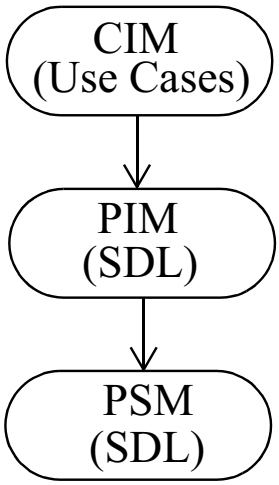

Figure 2: Model-driven design

There is a strong relationship between SDL and UML 2.0. For instance, the same concept of "system" is used, UML active classes are related to SDL process types, UML active components are related to SDL process instances, and UML ports and connectors are related to SDL gates and channels, respectively. This makes UML 2.0 an alternative candidate for model-driven design of ambient intelligence systems. We have chosen SDL because of our long-standing experience with the language, the early availability of commercial tools, the available reuse support, and its formal semantics, and believe that it is an excellent specification technique for model-driven design.

To support the process steps from CIM over PIM to PSM, transformation heuristics, SDL design components and patterns can be used. *Transformation heuristics* capture informal design guidelines, for instance, how to derive a partial PIM architecture from a set of use cases, and how to refine the PIM architecture into a PSM architecture. *SDL design components* are formally specified, ready-to-use solutions, which are selected from a library (called *package* in SDL) and composed [3]. *SDL design patterns* [4] are formalized, generic solutions to specific design problems, which are selected from a pattern pool, adapted to their specific application context, and composed. An incremental reuse-based development process model incorporating both SDL design components and

patterns and a detailed case study can be found in [3].

In Figures 3 through 7, the model-driven design steps are illustrated by several excerpts from the ABT. Figure 3 shows an excerpt of a use case describing functionality of the ABT, which is decomposed into a cyclist and a trainer system associated with the human cyclist and the human trainer, respectively. The cyclist system collects sensor data (e.g., pulse rate, speed), which are displayed to the human cyclist (function *update cyclist display*). The sensor data are transmitted to the trainer system (function *distribute cyclist status*). Commands of the human trainer (e.g., speed or position changes) are processed and sent by the trainer system and displayed by the cyclist system (function *update cyclist display*).

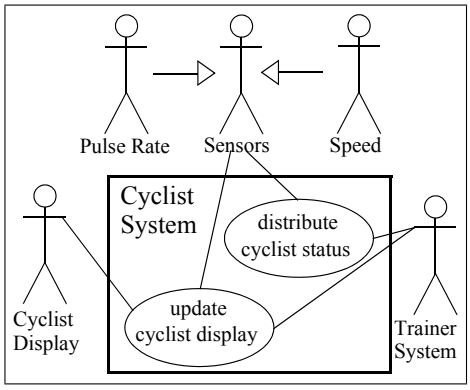

Figure 3: CIM: use case (excerpt)

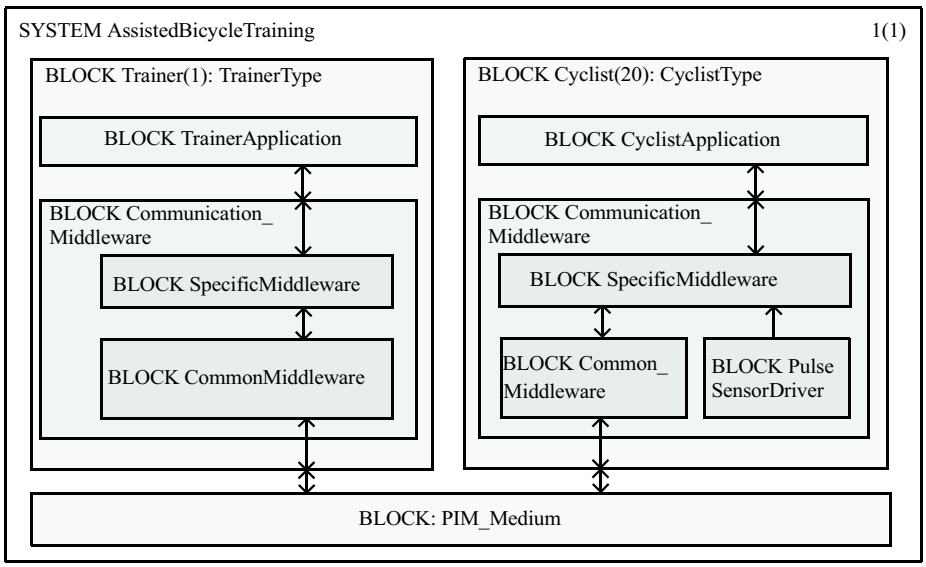

Figure 4: PIM: system architecture

Figure 4 is an SDL overview diagram, describing the upper layers of the PIM architecture. The diagram shows one trainer and 20 cyclists, decomposed into an application layer, a communication middleware, and a common medium. The communication middleware is further decomposed into an application specific middleware, which supports, for instance, the periodic transmission of cyclist stati to the trainer node, and the common middleware, which is responsible for routing, connection handling, and loss control.

The common medium is still platform-independent at this stage, but includes sufficient detail to capture commonalities of real media, such as message loss and network topology.

Some parts of the PIM architecture can be derived from the CIM use cases, based on informal transformation heuristics:

- The PIM is designed as a closed SDL system, incorporating both the system under development and its environment.
- For every system of the use cases, introduce an SDL block on SDL system level. For instance, the system `Cyclist` is represented by the block `Cyclist` in Figure 4.
- If there are interacting systems in the use case, add a communication middleware and a common abstract medium.
- For each sensor and actuator, add a corresponding driver and stub, respectively.

Our incremental development process ensures that after each increment, the functionality of the intermediate PIM can be analysed, using the SDL simulator [12]. In addition, SDL design components and patterns foster a reuse-based system design.

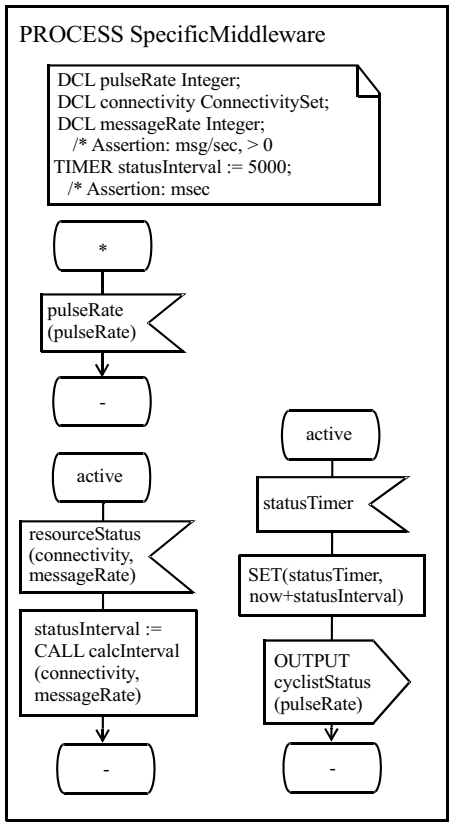

Figure 5: PIM: behavior excerpt (simplified)

AmI systems operate under scarce and/or dynamically varying resources. Furthermore, the application context may be subject to change. These characteristics require *adaptive mechanisms* that are to be *integrated across layers*. For instance, the number of cyclists that can deliver status data to the trainer - an application parameter - may be derived from the current network connectivity. This parameter can in turn be determined by the communication middleware and depends, among others, on the routing protocol (see below). Network connectivity may be used to upgrade or downgrade the quality of service, depending on the status message rate that is currently supported by the network. For good accuracy of the cyclist stati, the status interval should be as short as possible. Based on the number of cyclists that are within reach of the trainer, the interval can be calculated such that the available status

message rate is fully exploited.

Figure 5 shows a simplified excerpt of the behavior of the SDL process `SpecificMiddleware`, which forms part of the SDL block of the same name. The excerpt consists of three transitions. The first transition records the current pulse rate of the cyclist, measured and forwarded by a pulse sensor. The second transition takes the current resource status, consisting of the current connectivity and the supported status message rate, and determines the current status interval. This interval is used by the third transition, which outputs the current cyclist status to the trainer, to set the timer for the subsequent status message. This simple adaptive algorithm ensures that the status is always sent at the optimal rate, to avoid overloading the medium.

Another important design decision is the choice of a suitable broadcast algorithm as part of the common middleware. We have designed protocols for local broadcast, and one for global broadcast that is based on NXP/MPR (Neighbor eXchange Protocol/Multi Point Relay) [11], a sophisticated selective flooding protocol, which examines all neighbours within 2 hops distance and by the use of this information calculates the minimum necessary forwarding nodes. Depending on the application context and the network topology, a suitable broadcast protocol is selected and incorporated into the common middleware.

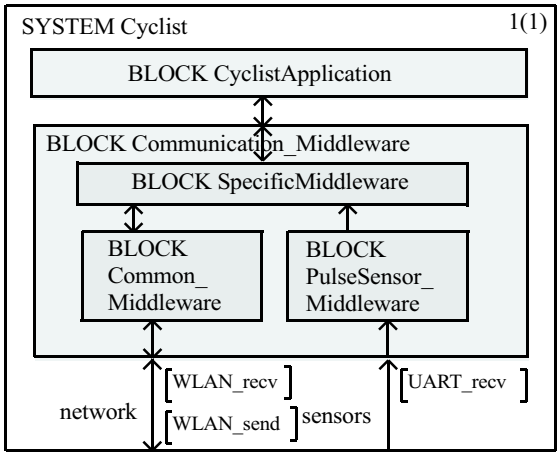

Figure 6: PSM: system architecture (excerpt)

Once the PIM is completed, the transformation into the PSM is carried out, again supported by SDL design components and patterns. In addition, the following transformation heuristics may be applied:

- Every SDL block that is to be mapped to a separate node is transformed into an SDL system. SDL systems are deployment units, and can be compiled separately.
- Remove the logical medium, and add - for each communication technology - corresponding signals to the environment. In addition, encoding and decoding routines may be needed.
- Every driver/stub is replaced by an SDL block, specifying the hardware-specific communication middleware, and a channel to the environment.

Following these heuristics, several transformations are performed in the ABT case study:

- The SDL block `Cyclist` is transformed into a single SDL system `Cyclist`. This system may later be installed on several nodes.
- The interface to the `PIM_Medium` is replaced by an interface to an existing communication technology, as shown in Figure 6 for wireless LAN. As a consequence, the behavior of the common middleware is augmented by encoding and decoding routines.
- The SDL block `PulseSensorDriver` is replaced by a block `PulseSensorMiddleware`. This means that the abstraction of the PIM, where pulse rates are determined by some function `pr`, is refined into an external sensor application and an interface, communicating with the external device via UART. The behavior before and after the refinement is shown in Figure 7.

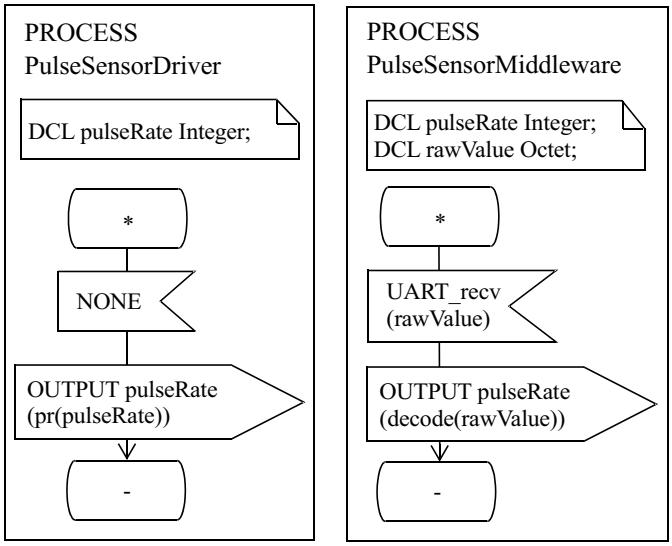

Figure 7: PIM and PSM: behavior refinement (simplified)

3.2 Model-Driven Implementation

In Figure 8, the late phases of the model-driven development process model are shown. Starting point is the PSM of the design phase (see Section 3.1), specified with SDL. From SDL specifications, it is possible to generate code in two steps. In the first step, intermediate code in languages such as C or C++ can be compiled. This code can be executed in different runtime environments (e.g., TAU Simulator [12]) and is therefore referred to as *Runtime-Independent Code (RIC)* in Figure 8. The commercial TAU tool set [12] provides two SDL-to-C code generators. *Cadvanced* is a full-scale compiler, supporting most constructs of SDL including dynamic creation of SDL processes and stateful procedures.

Cmicro implements a reduced subset of SDL and is targeted towards embedded systems.

To be executed on a specific target system, the RIC is compiled to machine code (*Runtime-Specific Code (RSC)*, see Figure 8), using a platform-specific C-compiler. To execute the RSC, an SDL engine for the target system is required in addition. The SDL engine comprises all functionality that is necessary to initialize and execute the SDL system, e.g., to build up the system structure, to select, schedule, and execute fireable transitions, and to transfer signals between SDL processes. SDL engines for Windows and Unix platforms are available from the TAU tool set.

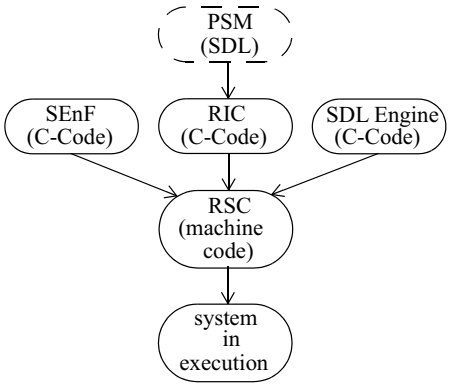

Figure 8: Model-driven implementation

To implement open SDL systems, i.e., systems interacting with their environment, one or more interfaces satisfying the semantics of the SDL signalling mechanism are needed. In general, the environment interface depends on a variety of aspects, such as the type of interaction supported by the environment (message passing, method invocation), the interaction formats, the operating system, and the communication service (connection-oriented, connection-less, addressing). According to the PSM, SDL systems may communicate via WLAN, Bluetooth, to name a few communication technologies, or with input and output devices such as joysticks, respectively. When using the TAU tool set, extensive manual, error-prone coding steps are required to supply these interfaces (also called *environment functions*).

To avoid manual coding entirely, we have developed a generic, specification independent environment package called *SDL Environment Framework (SEnF)* that is placed between the SDL system and the environment. The RIC of the open SDL system provides an interface description of the environment in terms of signal lists. Each signal to the environment can be associated with one specific communication technology or input/output device, for which our SDL Environment Framework automatically adds ready-to-use code modules to the RIC. The code modules offer support for various operating systems and hardware platforms. Thus, the combination of the SDL environment framework and SDL engine close the semantic gap between the abstract model (SDL specification of the PSM) and the implementation platform. Currently, SEnF covers the communication technologies IEEE-802.11a/b/g (WLAN), IEEE-802.15.1 (Bluetooth), RS-232 (UART), I^2C, the input/output devices web cam, joystick, several sensors/actuators (via I^2C), LEDs, and the operating systems Windows NT/2000/XP, Linux.

Figure 9 shows an ABT system configuration with n cyclists and one trainer, communicating via WLAN 802.11b and operating under Linux and Windows, respectively. In

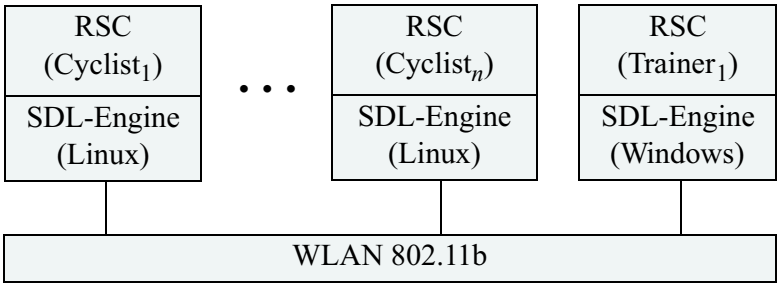

Figure 9: RSC: System configuration

addition, there is communication with the pulse sensor via UART, and with a PDA (see below). Please note that the same RIC is used to compile the RSCs for different target systems. Also, different operating systems are combined into one distributed system.

The prototype cyclist system is depicted in Figure 10. On the carrier, the embedded PC *Arbor Technology Em104P-i6023* (with WLAN stick Netgear MA-111, Bluetooth adapter D-Link DBT-120, and UART interface), pulse rate receiver, and batteries (Lithium-Polymer, 1500 mAh) are mounted. A PDA (Acer n-30) showing the current driver status (e.g., pulse rate, actual speed) and the trainer orders (e.g., required speed, required position changes) is attached to the handlebar. Communication between embedded PC and PDA is via Bluetooth. Finally, the cyclist carries a pulse rate transmitter. The trainer system (not shown here) is installed on a laptop, with a sophisticated graphical interface to monitor and direct the training situation. So far, we have equipped 3 bicycles with the cyclist system, as shown in the figure, and have successfully run several training sessions.

Figure 10: Equipped bicycle

3.3 Model-Driven Performance Simulation

In several case studies, we have demonstrated that the PSM can also be used as the starting point for model-driven performance simulations. In Figure 11, the corresponding phases of the process model are shown. Similar to model-driven implementation, the RIC and RSC are generated and combined with the SDL Engine and the SDL Environment Framework. In addition, our network simulator *ns+SDL* [6] is linked to the machine code, controlling the execution of the system under simulation.

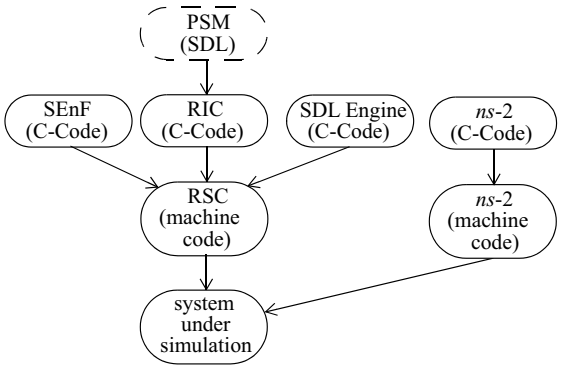

Figure 11: Model-driven performance simulation

The original network simulator *ns-2* [13], developed at the Information Sciences Institute of the University of Southern California, is a frequently used tool for the simulation of computer networks, composed of transport, routing, and multicast protocols over wired and wireless networks. *ns-2* is an event-based simulator, consisting of the simulation scheduler and a set of simulation components that can be configured in a flexible way, yielding an executable simulation system. The simulation scheduler globally controls all events during a simulation.

In [6], we have extended *ns-2* to incorporate components that are generated from SDL specifications, using the aforementioned code generator TAU Cadvanced. This enables the developer to use code that is generated from the PSM for performance simulations of networked systems. Our extension consists of several *ns-2* simulation components replacing predefined simulation functionalities, an adapted SDL Engine for the interaction between *ns-2* and an SDL system, and an environment package for SDL systems that forms part of the SDL Environment Framework (SEnF). An important advantage of our solution is that the same SDL-to-C code generator is used to compile the runtime-independent code (RIC) from the same PSM. This increases confidence that the results of the performance simulation hold for the system in execution.

Figure 12 shows an ABT simulation configuration with 20 cyclists and one trainer, communicating via a simulated WLAN 802.11b wireless LAN with a range of about 200m. The cyclists are riding one behind the other, followed by the trainer. According to the mobility model, positions and distances of cyclists changes during the ride. While this has no consequences on connectivity between nodes in most cases, due to the wide range of WLAN, there are two situations where the field and the network are partitioned. At simulation times t1 = 100 sec and t2 = 530 sec, there is a gap of 200 m betweeen nodes 9 and 10.

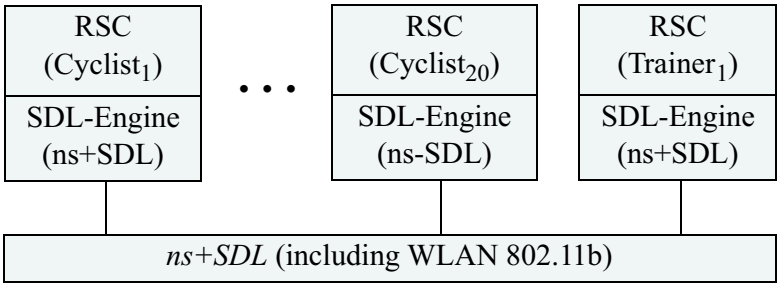

Figure 12: RSC: Simulation configuration

In our model-driven simulations, we have examined two system aspects. The first aspect concerns the comparison of local and global broadcast communication (see Figure 13). In the local broadcast case, the connectivity decreases to around 50%, when the field is partitioned. The selective flooding protocol NXP/MPR improves this situation substantially, providing for almost full connectivity during the entire simulation. Reduced connectivity only occurs for short periods of time, and is due to frame collisions that prevent neighbors to receive the updated networks status.

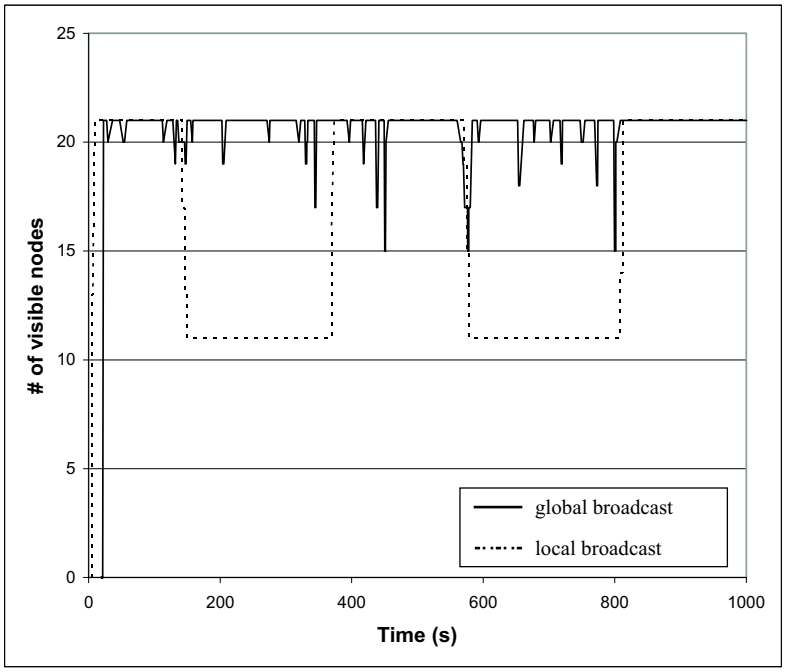

Figure 13: Comparison of local and global (NXP/MPR) broadcast

The second aspect concerns the benefits of the algorithm to adapt the status message rate to the current number of cyclists in the group. Simulation results are shown in Figure

14 for local broadcast communication. In the non-adaptive case, the maximum number of cyclists in the group, i.e., 30 (see Section 2), is used to determine the (constant) status message rate as supported by the network and observed by the trainer, which is 7 per second. Since the actual group size is only 20, this leads to an actual status message rate of 5 per second when all members of the group are within reach of the trainer. This rate drops to 2 per second during periods of network partitioning.

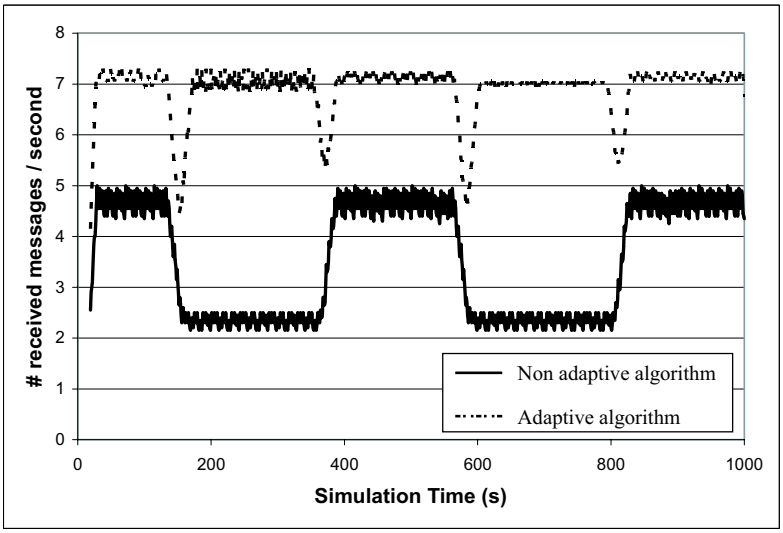

Figure 14: Benefits of the status message rate adaptation (local broadcast)

In the adaptive case, the actual number of cyclists is determined and updated dynamically. In the simulation, the actual number of cyclists in the group allows for 7 status messages per second at the beginning. When the group is split, there is a short drop down to 5 status messages per second before the updated number of cyclists leads to a reduced status message interval of the cyclists within range of the trainer, and therefore to the maximum rate of 7 per second. Interestingly, there is another drop when the field of cyclists fuses. This can be explained by the fact that the previous field members reduce their message rate immediately (due to the larger group size), but the new field members start their status message transmission only after their status interval has expired for the first time.

Since the performance simulation is model-based, it is straightforward to replace the WLAN simulation component by other technologies, say, ZigBee^{TM} or Bluetooth, and to study the above system aspects. Also, it is straightforward to replace the broadcast protocol, or other communication components, or to modify the adaptation algorithm. For this, it is important that the platform-independent model is sufficiently detailed. We have shown that this is feasible even in the context of ambient intelligence systems, which are constrained by resources and environmental conditions.

4 Summary and Outlook

In this paper, we have built on and refined the idea of OMG's MDA to place the abstract, formal system model in the center of the development activity, using ITU-T's SDL as design language. We have presented a comprehensive, holistic model-driven approach that covers the entire development trajectory and incorporates performance simulation:

- With our tool chain, we replace manual coding steps entirely. Using our generic, specification-independent environment package SDL Environment Framework, we can generate interface routines for different combinations of operating systems (e.g., Linux, Windows), communication technologies (e.g., WLAN, UART, Bluetooth) and IO devices (e.g., LEDs, web cams, joysticks).
- We have demonstrated that the platform-specific model can be used as a common code base for the generation of simulation and production code. This is an important step forward, as it is no longer necessary to have a separate, hand-coded simulation system that needs to be kept consistent with the system model. Furthermore, the same compiler has been used to generate the simulation and production code. Both measures increase confidence that the results of the performance evaluation hold for the networked system in operation.
- To provide evidence for the feasibility and usefulness of our approach, we have conducted a substantial case study of an ambient intelligence system, the Assisted Bicycle Trainer. Here, the focus has been on the peculiarities of AmI systems, in particular, scarce and dynamic resource situations, adaptivity, and cross-layer integration. These aspects have already been incorporated in the early development phases, and have enabled a performance assessment that was based on the platform-specific model, using *ns+SDL*, an extension of *ns-2* that we have developed for this purpose.

Future work in several directions can be envisaged:

- The SDL Environment Framework (SEnF) can be extended to support further hardware platforms, communication technologies, and IO devices. We are currently working on interface routines for MICAz [2], a tiny, wireless measurement system communicating via IEEE 802.15.4 ZigBee^{TM}, in order to replace the embedded PC of the Assisted Bicycle Trainer.
- Further case studies will be conducted to evaluate, assess, and improve our model-driven approach. This concerns the investigation of sophisticated adaptive algorithms, cross-layer integration of quality-of-service functionality, as well as the model-driven development of AmI systems for assisted living. Further work is

foreseen to improve the Assisted Bicycle Trainer, by taking additional cyclist data into account, and by cooperating with cyclist training teams and sports physicians.

5 Acknowledgements

We gratefully acknowledge the financial support of the Deutsche Forschungsgemeinschaft (DFG), and of the Interdiscplinary Research Center "Ambient Intelligence" at the University of Kaiserslautern.

References

[1] M. Book, S. Beydeda, and V. Gruhn. *Model-driven Software Development*. Springer, 2005.

[2] Crossbow. Micaz wireless measurement system. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf`.

[3] I. Fliege, A. Geraldy, R. Gotzhein, T. Kuhn, and C. Webel. Developing Safety-Critical Real-Time Systems with SDL Design Patterns and Components. *Computer Networks, Elsevier*, 2005. (forthcoming).

[4] R. Gotzhein. Consolidating and Applying the SDL-Pattern Approach: A Detailed Case Study. *Information and Software Technology*, 45:727–741, 2003.

[5] International Telecommunications Union. *Specification and Description Language (SDL)*. ITU-T Recommendation Z.100, August 2002.

[6] T. Kuhn, A. Geraldy, R. Gotzhein, and F. Rothländer. ns+SDL - The Network Simulator for SDL Systems. In A. Prinz, R. Reed, and J. Reed, editors, *SDL 2005*, Lecture Notes in Computer Science (LNCS) 3530, pages 103–116. Springer, 2005.

[7] L. Litz, N. Wehn, and B. Schürmann. Research Center "Ambient Intelligence" at the University of Kaiserslautern. In *VDE Kongress 2004*, volume 1, pages 19–24, Berlin/Germany, 2004. VDE Verlag, ISBN 3-8007-28273.

[8] J. Miller and J. Mukerji, editors. *MDA Guide Version 1.0.1*. OMG, 2003.

[9] Object Management Group. Unified Modeling Language 2.0 Infrastucture Final Adopted Specification. `http://www.omg.org/cgi-bin/doc?ptc/2003-09-15`, 2004.

[10] E. Sherratt, editor. *Telecommunications and beyond: The Broader Applicability of SDL and MSC*. Lecture Notes in Computer Science (LNCS) 2599. Springer, 2003.

[11] T. Sonntag. Optimized flooding for a voice radio application as example. (in german), project thesis, University of Kaiserslautern, Computer Science Department, 2005.

[12] Telelogic AB. Telelogic Tau Generation 1. http://www.telelogic.com/products/tau/index.cfm.

[13] The Network Simulator ns-2. http://www.isi.edu/nsnam/ns. Information Sciences Institute, University of Southern California.