Model-driven by SDL – Improving the Quality of Networked Systems Development
(Invited Paper)

Reinhard Gotzhein

Networked Systems Group, University of Kaiserslautern,
D-67663 Kaiserslautern, Germany

gotzhein@informatik.uni-kl.de

ABSTRACT. *In this paper, we outline our holistic, model-driven approach to the development of networked systems, with SDL as modeling language. With this approach, distributed applications and specialized communication protocols can be developed together. The specification of design models is supported by several reuse methods: SDL patterns, SDL components (in particular, micro protocols), and SDL framework. Implementations are automatically generated from design models, and automatically instrumented to interface with different operating systems and communication technologies. The same design models are also used for performance simulations, which increases confidence in simulation results. The approach has been applied to develop various communication protocols for mobile ad-hoc networks, networked systems in the domains of ambient intelligence and building automation, and telecommunication systems.*

RÉSUMÉ. *Dans ce papier, nous donnons un aperçu global de notre approche basée sur les modèles pour le développement de systèmes distribués. Elle utilise SDL comme langage de modélisation. Avec cette approche, des applications distribuées ainsi que des protocoles de communication spécialisés peuvent être développés simultanément. La spécification des modèles est appuyée par plusieurs méthodes de réutilisation: les patrons SDL (SDL patterns), les composantes SDL (en particulier, les micro-protocoles), et un cadre de travail SDL (SDL framework). Des implémentations sont générées automatiquement à partir de modèles de conception, qui sont aussi utilisées pour la simulation des performances, ce qui augmente la confiance dans les résultats de simulation. L'approche a été appliquée pour le développement de différents protocoles de communication pour réseaux mobiles ad-hoc, de systèmes distribués dans les domaines tels que intelligence ambiante et bâtiments intelligents, et systèmes de télécommunications.*

KEYWORDS: *model-driven development, networked systems, communication protocol, design component, design pattern, design framework, development tools, performance simulation, SDL*

MOTS-CLÉS : *développement basé sur les modèles, systèmes distribués, protocoles de communication, composantes de conception, patrons de conception, cadre de travail conceptuel, outils de développement, simulation de performance, SDL*

2e soumission à *Notère 2007*, le 15 fèvrier 2007.

1. Introduction

Over the past decade, we have witnessed the entering of modeling techniques into industrial software development. Part of this success story is due to the UML family of notations [UML04], which covers several development phases, and is supported by industrial tools. Another part of this process is the turning towards Model Driven Development (MDD) [BOO 05], a software engineering approach that places the abstract, formal system model in the center of the development activity. The objective is that models guide and direct all development activities, ranging from system design over code generation and deployment to system maintenance, resulting both in quality improvements and productivity increases.

Model-driven development is promoted by the Object Management Group in the Model Driven Architecture™ (MDA) initiative [MIL 03], and supported by the UML 2.0 [UML04]. To focus on specific system aspects, MDA introduces three kinds of viewpoints. The computation independent model (CIM) represents the view of the domain practitioner, omitting all details of the system structure. The platform independent model (PIM) defines the view of the system developer by providing structural and functional details, suitable for use with a number of different platforms of similar type. The platform specific model (PSM) augments the PIM by details of how the system uses a specific platform. These models direct the course of understanding, design, construction, deployment, operation, maintenance, and modification of a system. Important concerns of MDA are the transformations from CIM to PIM and from PIM to PSM, and the automatic code generation from models.

To define system views, we have adopted SDL, the Specification and Description Language [ITU 02], standardized by the International Telecommunications Union (ITU). SDL is a sophisticated formal design language for the specification of networked systems and communication protocols, with graphical syntax, data types, structuring concepts, support for reuse, and commercial tool environments. Major industrial application areas of SDL are telecommunication and automotive systems [SHE 03].

In this work, we outline our holistic, model-driven approach to the development of networked systems and communication protocols, with SDL as modeling language, combining some of our research results of the last decade. We start with the design phase in Section 2, addressing the definition of CIM, PIM, and PSM, and in particular focusing on the transformation between these views by means of formalized reuse approaches. Section 3 is devoted to model-driven implementation of SDL designs, using a complete tool chain for automatic code generation. In Section 4, we elaborate on the use of SDL designs for performance simulations, supported by several simulators on different levels of detail. We give conclusions and an outlook in Section 5.

2. Model-driven Design with SDL

In this section, we outline the early phases of our model-driven development process, which are fully in line with the OMG MDA (see Figure 1). The computation independent model (CIM) is expressed by message scenarios, specified with MSC [ITU 01], and informal text. For the specification of the platform independent and platform specific models (PIM and PSM), we use SDL, ITU-T's Specification and Description Language [ITU 02]. To support the process steps from CIM over PIM to PSM, we apply several structuring and reuse methods, which will be surveyed in this section.

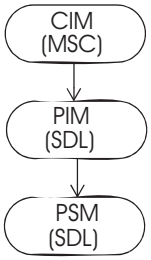

Figure 1. *Model-driven design*

2.1. *SDL Patterns*

SDL patterns [GEP 97] are formalized, generic, possibly distributed solutions to specific, recurring design problems of networked systems and communication protocols, with SDL as design language. They combine the traditional advantages of design patterns - reduced development effort, quality improvements, and orthogonal documentation - with the precision of a formal design language for pattern definition and pattern application. SDL patterns support reuse-driven design, raise the vocabulary of the system engineer to a problem-oriented level, assist the discovery and exploitation of commonalities, and lead to well-justified, high quality designs.

The SDL pattern approach [GEP 97, GOT 03a] consists of the *SDL pattern design process*, a notation for the description of generic SDL fragments called *PA-SDL* (Pattern Annotated SDL), a *template* and *rules* for the definition of SDL patterns, an *SDL pattern pool* for networked systems and communication protocols, and *tool support* [DOR 04]. The pattern pool can be seen as a repository of experience from previous projects that has been analyzed and documented for reuse purposes.

When SDL patterns are applied, they are selected from the pattern pool, adapted to their specific application context, and composed. The basis for pattern selection is an analysis model, consisting of a set of concrete message scenarios, specified with MSC. Here, the CIM, which also is a collection of message scenarios, forms the starting point. Structural refinements lead to further scenarios, and to the selection of additional applicable design patterns. When selected, SDL patterns are adapted to a specific context by replacing generic parts of the pattern definition, yielding pattern instances. This adaptation is guided by rules and relationships between the concrete and generic analysis models. Finally, the pattern instance is embedded into the context specification.

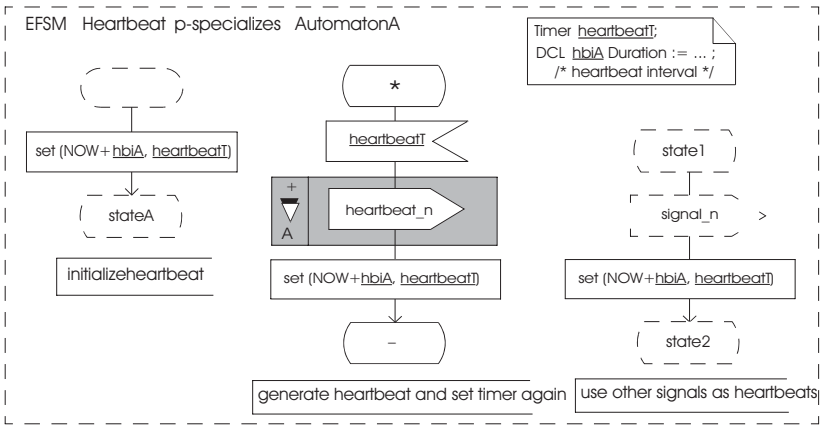

Figure 2. *SDL design pattern* HEARTBEAT*: SDL Fragment*

An SDL pattern is more than just an incomplete piece of SDL specification. As detailed, for instance, in [GOT 03b], a pattern definition provides information about intention, motivation, structural aspects, message scenarios, refinements, cooperative usage, generic SDL fragment, embedding rules, etc. To give an idea of a pattern definition, we show part of the generic SDL fragment of the HEARTBEAT pattern (Figure 2, [FLI 05]). The EFSM Heartbeat describes the heartbeat functionality that is used together with a watchdog (see Section 2.2) to detect component failures in distributed systems. During initialization, the timer heartbeatT is set to the duration of hbiA (heartbeat interval): after its timeout, a non-empty sequence of heartbeat signals is generated and propagated, with the timer being set again. Other signals may also be interpreted as heartbeats, a method to save bandwidth. To apply the pattern, a context AutomatonA is identified and modified, by redefining the start transition and adding a new transition as defined in Figure 2. Adaptations are constrained by elements of PA-SDL (see, e.g. [FLI 05]).

The heartbeat pattern has been instantiated in the design of the remote control of the airship "James Blimp" (see Figure 3). The airship is operated remotely by a pilot on the ground, who is equipped with joystick attached to his laptop. During regular operation, control values are communicated via WLAN from the laptop to the airship periodically, as parameters of the heartbeat signal. If for some reason, e.g., control system failure or subsequent heartbeat losses, control values are not received for a predefined interval, the airship changes into a fail-safe state, switching off all propellers.

SDL patterns have been applied to the development and reengineering of a variety of communication protocols and distributed systems. This includes the airship presented above [FLI 05], the Internet Stream Protocol ST2+ [Röß 98], quality-of-service protocols on top of the CAN field bus [GEP 98], and the customized communication system of a light control application [SCH 03]. Over a period of 10 years, the SDL

Figure 3. *The airship "James Blimp" in operation*

pattern approach has been consolidated and applied in industry, in particular, in UMTS RNC call processing development, where a quantitative assessment of cost reduction due to quality improvements has been conducted [GRA 03].

2.2. *SDL Components*

SDL components [FLI 05] are ready-to-use, self-contained design solutions, which are selected from a component library and composed. In comparison to SDL patterns, they are more specific, and can be expressed as a set of SDL agent types. In [GOT 02] and [GOT 03b], we have extended the idea of SDL components by the concept of *micro protocol*. A micro protocol is a *distributed* component providing one single protocol functionality - e.g., flow control, loss control, resource reservation - that is not decomposable into smaller self-contained protocol units. Due to its distributed nature, interaction between protocol entities is part of the micro protocol definition.

The concept of micro protocol is illustrated in Figure 4, which shows a micro protocol realized by two micro protocol entities and a collaboration that establishes a causal relationship between protocol events e1 and e2. Micro protocol entities interact (directly or indirectly) with their environment, consisting of further (micro) protocol entities, service providers, and service users. A micro protocol can be defined operationally by specifying architecture, behavior of protocol entities, and data formats.

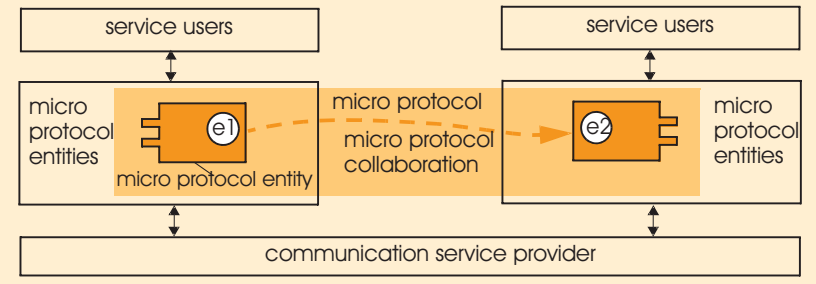

Figure 4. *The concept of micro protocol*

Since entities of a given micro protocol are typically instantiated on different network nodes, a micro protocol can be seen as a distributed component.

The definition of SDL components is supported by the language concepts *SDL package* and *SDL type*. An SDL package is a collection of type definitions, and used here to encapsulate all SDL types of a given component. When a component is to be composed, the SDL package defining the component is imported, and the type definitions are instantiated in the embedding design context. This especially applies to micro protocols, where multiple component instances are created on different nodes.

An SDL component definition provides information about intention, interface, architecture, message scenarios, used SDL packages, and the SDL design itself. Figure 5 (see [FLI 05]) shows an excerpt of the SDL design, the process type definition of Watchdog. The component has three transitions. During the start transition, the heartbeat interval is set. Once active, the watchdog monitors heartbeats (see Section 2.1.), and triggers another component if this signal is not received for the heartbeat interval hbiW.

The definition is syntactically complete and self-contained, however, it still leaves room for some adaptations, exploiting SDL language features. For instance, the start transition may be specialized to redefine the heartbeat interval. Also, new transitions may be added if other signals are to be interpreted as heartbeats. The watchdog component has been instantiated in the design of the remote control of the airship "James Blimp" (see Figure 3).

SDL components and/or micro protocols have been specified and instantiated in the development and reengineering of several distributed applications and communication protocols. In [GOT 03b], we have reengineered SNMPv1 (Simple Network Management Protocol). In [FLI 05], watchdog and heartbeat components have been devised and instantiated. In this volume, the application of the micro protocol approach to the engineering of complex routing protocols for ad-hoc networks is presented [FLI 07]. Another application can be found in [KUH 06a], where the functionalities of a MAC layer for Berkeley motes have been defined as micro protocols.

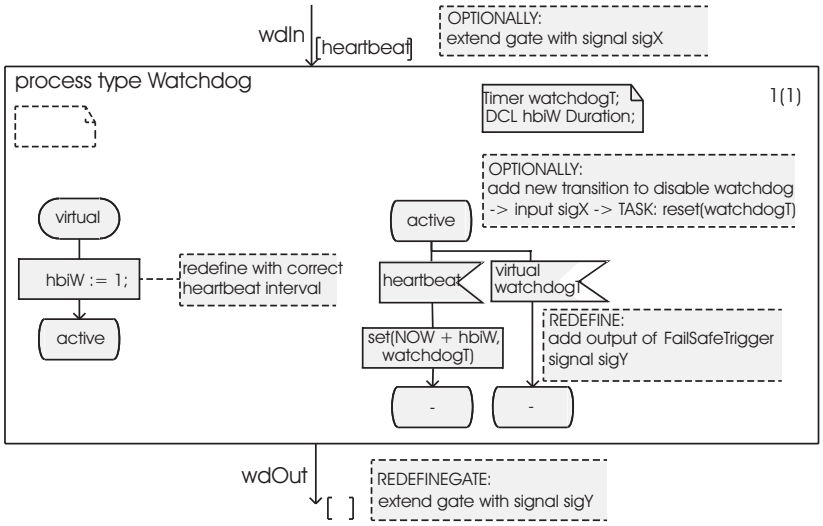

Figure 5. *SDL micro protocol* Watchdog*: SDL design*

2.3. *SDL Framework*

In a narrower sense, an *SDL framework* [BRÆ 03] is the skeleton of a (distributed) system, to be completed by the system designer. To define frameworks, the capability of SDL to encapsulate object structures and default behavior in frameworks defined by SDL system types is exploited. Specific applications can then be obtained by defining subtypes of framework system types, and by redefining virtual types.

In [FLI 04], we have generalized the concept of framework, and have applied it in the context of micro protocols. To be open to different structuring principles when designing communication systems, we have defined *micro protocol framework* as a set of general principles and rules for the composition of micro protocols. This means that the system skeleton is not provided in advance, but is tailored in accordance with the framework.

With the operators provided by the micro protocol framework, micro protocols can be composed to form protocols with more complex functionalities, called *macro protocols*. Micro protocol entities may be composed in sequence, yielding a signal processing pipeline. Also, they may be composed concurrently, hierarchically, or in an interleaving manner. In cases of stronger functional dependencies between micro protocol entities, these generic forms of composition are augmented by additional synchronization that refers to internal, problem specific protocol aspects.

A particular architecture that may be derived from the generic micro protocol framework is illustrated in Figure 6. Here, the micro protocol of Figure 4 is composed with further micro protocols in two steps, yielding (macro) protocols with more

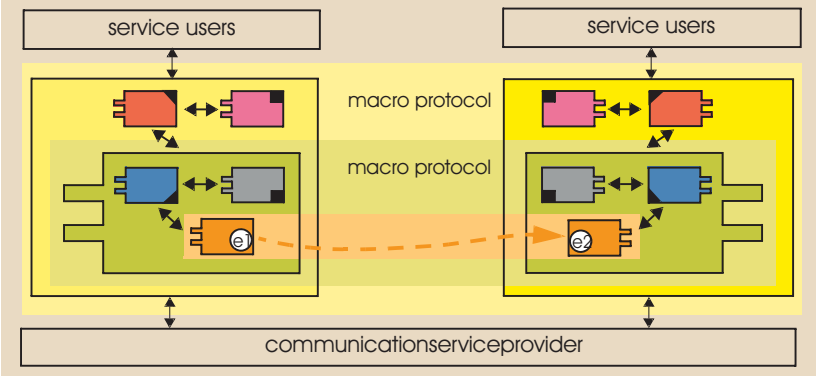

Figure 6. *A micro protocol architecture*

complex functionalities. Entities belonging to the same micro protocol are marked by identical graphical shapes. Depending on the applied composition, different kinds of architectures including layered architectures and functional architectures [ZIT 93] can be obtained.

3. Model-driven Implementation with SDL

Starting point for model-driven implementation is the platform-specific model of the design phase, specified with SDL (see Figure 7). 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 and is therefore referred to as *Runtime-Independent Code* (*RIC*) in Figure 7. The commercial TAU tool set [TAU] provides two SDL-to-C code generators for this purpose. *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. In addition, we have developed our own SDL-to-C code generator *ConTraST* [FLI 06b], which can be configured to support different SDL profiles [GRA 06]. Based on the intermediate representation of SDL specifications, ConTraST also generates the documentation of micro protocols.

To be executed on a specific target system, the RIC is compiled to machine code (*Runtime-Specific Code* (*RSC*), see Figure 7), 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, and from ConTraST.

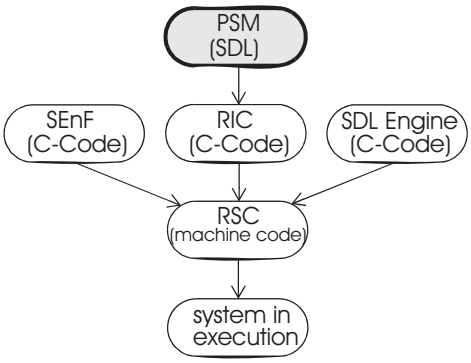

Figure 7. *Model-driven implementation*

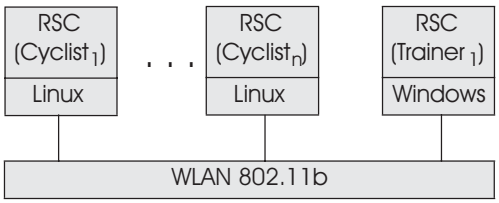

Figure 8. *RSC: System configuration*

To implement open SDL systems, i.e., systems interacting with their environment, one ore more interfaces satisfying the semantics of the SDL signaling mechanism are needed. In general, the environment interface depends on a variety of aspects, such as the type of interaction supported by the environment (e.g., message passing, method invocation), the interaction formats, the operating system, and the communication service (e.g., connection-oriented, connection-less, addressing). According to the PSM, SDL systems may communicate via WLAN, UART, Bluetooth, or Zigbee, to name a few communication technologies, or with input and output devices such as LEDs, joysticks, net cams, sensors, and actuators, respectively. When using the TAU tool set, extensive hand-coding steps are required to supply these interfaces (called *environment functions* in the context of TAU). To avoid hand-coding entirely, we have developed a generic, specification independent library of interfacing routines, called *SDL Environment Framework* (*SEnF*). Based on configuration information supplied by TAU, interfacing routines for different combinations of operating systems, communication technologies, and I/O devices are automatically determined and added to the generated code.

Figure 8 shows an ABT system configuration with n cyclists and one trainer, communicating via WLAN 802.11b and using Linux and Windows, respectively. In addition, there is communication with the pulse sensor via UART, and with a PDA. Please

Figure 9. *Equipped bicycle*

note that the same RIC is used to compile the RSC for different target systems. Also, different operating systems are combined into one distributed system.

The prototype cyclist system is depicted in Figure 9. On the carrier, the embedded PC (with WLAN stick, Bluetooth adapter, and UART interface), pulse rate receiver, and batteries are mounted. A PDA 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 user interface to monitor and direct the training situation.

A weight-optimized version of the cyclist system has been implemented on a MICAz platform, a low power wireless sensor mote produced by Crossbow Industries [CRO]. This mote consists of an embedded micro controller with 128 KB of Flash ROM and 4 KB of RAM. The PSM for the MICAz platform is very similar to the PSM of the embedded PC solution. While the PIM is identical for both platforms, the WLAN driver interface has been replaced by a CC2420 ZigBee interface. In addition, a specialized MAC layer has been added. Due to resource constraints, different SDL-to-C compilers and different SEnF interfacing routines have been used to generate the production code for the two platforms.

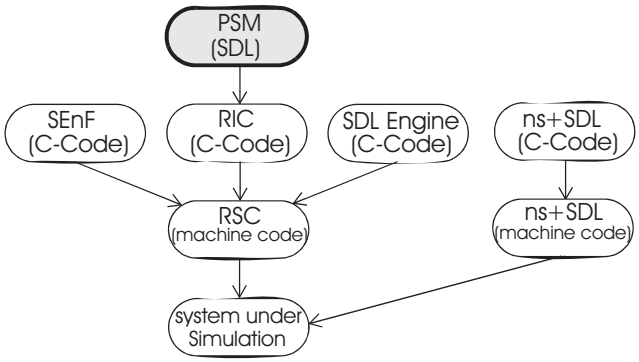

Figure 10. *Model-driven performance simulation*

4. Model-driven Performance Simulation with SDL

In several case studies [FLI 06a, KUH 06b, WEB 06], we have demonstrated that the platform-specific model can also be used as starting point for model-driven performance simulations. In Figure 10, the corresponding phases of the process model are shown. Similar to model-driven implementation, the RIC and RSC are generated and combined with SDL Engine and SDL Environment Framework. In addition, the network simulator *ns*+SDL [KUH 05] is linked to the machine code, controlling the execution of the system under simulation.

ns+SDL is an extension of the network simulator *ns*-2, an event-based simulator, developed at the Information Sciences Institute of the University of Southern California [ns2]. We have extended *ns*-2 to incorporate components that are generated from SDL specifications, using the aforementioned code generator TAU Cadvanced [TAU]. This enables the developer to use code that is generated from the PSM for performance simulations of networked systems. Our extension of *ns*-2 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 11 shows an ABT simulation configuration with 20 cyclists and one trainer, communicating via a simulated WLAN 802.11b wireless LAN with a range of more than $100m$. The cyclists are riding one behind the other, followed by the trainer. According to the mobility model, positions and distances of cyclists change 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

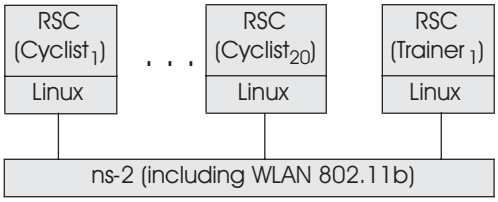

Figure 11. *RSC: Simulation configuration*

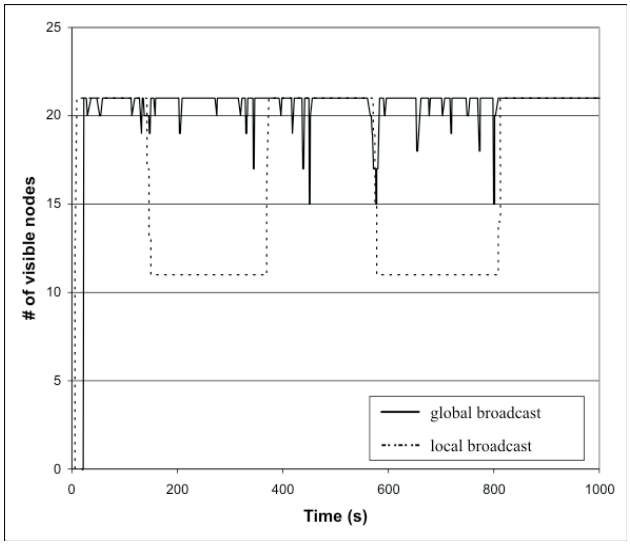

Figure 12. *Comparison of local broadcast and NXP/MPR*

the network are partitioned. At simulation times $t_1 = 100sec$ and $t_2 = 530sec$, there is a gap of about $100m$ between two groups of nodes.

In the simulated scenario, we have examined two system aspects. First, we have compared connectivity for local and global broadcast (see Figure 12). When using local broadcast, connectivity decreases to about 50% when the field is stretched. Global broadcast using selective flooding with NXP/MPR improves the situation substantially. Reduced connectivity only occurs for short periods of time, and is due to frame collisions that prevent neighbors to receive the updated network status.

The second aspect concerns the benefits of the algorithm to adapt the status message rate to the current number of cyclists in the group for best use of available bandwidth. Simulation results are shown in Figure 13. In the non-adaptive case, the maximum number of cyclists in the group, i.e., 30, is used to determine the (constant) status message rate as observed by the trainer, which is 7 per second. Since the actual group

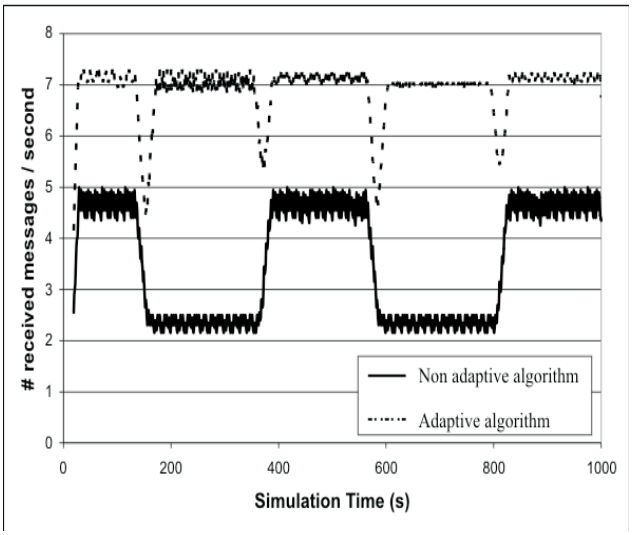

Figure 13. *Benefits of status message rate adaptation*

size is only 20, this leads to an actual status message rate of 5 per second when all members of the group are within range of the trainer. This rate drops to 2 per second during periods of network partitioning.

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 shorter 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), while 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 or Bluetooth, and to study the above system aspects. Also, it is straightforward to replace the routing 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.

5. Conclusions and Outlook

In this paper, we have outlined our holistic, model-driven approach to the development of networked systems, with SDL as modeling language, combining some of our research of the last decade. We have shown that these individual results are complementary and, taken together, support the entire process of model-driven development of distributed systems and communication protocols. This applies to design reuse approaches in the early development phases, and also to automatic code generation techniques in the late phases.

Certainly one of the most important results is that placing the model in the center of the development activity works well for distributed systems and communication protocols, and that formalized design methods are most useful. A complete tool chain is available to support design creation, design reuse, design simulation, design analysis, design documentation, performance simulation, production and simulation code generation, and environment interfacing. Some of these tools are available from a commercial tool provider; other tools have been developed in the course of our research. It is noteworthy to state that this tool chain has been applied successfully in a number of holistic system developments.

The holistic, model-driven approach outlined in this paper paves the way for future work on the engineering of distributed systems and communication protocols. We are currently studying adaptive routing algorithms for ad-hoc networks, i.e. algorithms that will adapt to node density, node movement, and traffic profiles. This requires sufficiently strong cross-layer integration of protocols and distributed applications. With SDL being well-suited for designing these system layers, and with our tool chain for model-driven performance simulations, we are capable of studying the effects of individual adaptation measures very early in the development process. Another research area is quality of service provision in ad-hoc networks, where adaptive protocols are crucial to cope with scarce resources, varying channel quality and traffic profiles, and node mobility. Finally, we are applying the concepts and tools presented in this paper to the development of an adaptive, hybrid MAC layer for wireless ad-hoc networks.

Acknowledgements. Over the past ten years, several researchers have contributed to the holistic approach outlined in this paper. This includes (in alphabetical order) Jörg Dorsch, Ingmar Fliege, Birgit Geppert, Alexander Geraldy, Rüdiger Grammes, Ferhat Khendek, Thomas Kuhn, Frank Rößler, Philipp Schaible, and Christian Weber. I thank all of them for their valuable contributions. The financial support of several funding organizations, in particular of the Deutsche Forschungsgemeinschaft (DFG), is gratefully acknowledged.

6. References

[BOO 05] BOOK M., BEYEDA S., GRUHN V., *Model-driven Software Development*, Springer, 2005.

[BRÆ 03] BRÆK R., MØLLER-PEDERSEN B., "Micro Protocol Design - The SNMP Case Study", SHERRATT E., Ed., *Telecommunications and Beyond - The Broader Applicability of SDL and MSC*, vol. 2599 of *LNCS*, Springer, 2003, p. 61-73.

[CRO] CROSSBOW, "MICAz Wireless Measurement System", `www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf`.

[DOR 04] DORSCH J., EK A., GOTZHEIN R., "SPT - The SDL Pattern Tool", AMYOT D., WILLIAMS A. W., Eds., *System Modeling and Analysis*, vol. 3319 of *LNCS*, Springer, 2004, p. 50-64.

[FLI 04] FLIEGE I., GERALDY A., GOTZHEIN R., SCHAIBLE P., "A Flexible Micro Protocol Framework", AMYOT D., WILLIAMS A. W., Eds., *System Modeling and Analysis*, vol. 3319 of *LNCS*, Springer, 2004, p. 224-236.

[FLI 05] FLIEGE I., GERALDY A., GOTZHEIN R., KUHN T., WEBEL C., "Developing Safety-Critical Real-Time Systems with SDL Design Patterns and Components", *Computer Networks*, vol. 49, num. 5, 2005, p. 689-706, Elsevier.

[FLI 06a] FLIEGE I., GERALDY A., GOTZHEIN R., JAITNER T., KUHN T., WEBEL C., "An Ambient Intelligence System to Assist Team Training and Competition in Cycling", MORITZ E. F., HAAKE S., Eds., *The Engineering of Sport 6, Vol. 1: Development for Sports, Springer Science and Business Media*, 2006, p. 103-108.

[FLI 06b] FLIEGE I., GRAMMES R., WEBER C., "ConTraST - A Configurable SDL Transpiler and Runtime Environment", GOTZHEIN R., REED R., Eds., *System Analysis and Modeling: Language Profiles*, vol. 4320 of *LNCS*, Springer, 2006, p. 216-228.

[FLI 07] FLIEGE I., GERALDY A., GOTZHEIN R., "Micro Protocol Based Design of Routing Protocols for Ad-hoc Networks", *7th International Conference on New Technologies of Distributed Systems (NOTERE 2007), Marrakesh, Morocco*, June 4-8, 2007.

[GEP 97] GEPPERT B., GOTZHEIN R., RÖSSLER F., "Configuring Communication Protocols Using SDL Patterns", CAVALLI A., SARMA A., Eds., *SDL'97: Time for Testing*, Proceedings of the 8th SDL Forum, Elsevier, 1997, p. 523-538.

[GEP 98] GEPPERT B., KÜHLMEYER A. K., RÖSSLER F., SCHNEIDER M., "SDL Pattern Based Development of a Communication Subsystem for CAN", BUDKOWSKI S., CAVALLI A., NAJM E., Eds., *Formal Description Techniques and Protocol Specification, Testing, and Verification*, Proc. of FORTE/PSTV'98, Kluwer Academic Publishers, 1998, p. 197-212.

[GOT 02] GOTZHEIN R., KHENDEK F., "Conception avec Micro-Protocoles", *Colloque Francophone sur l'Ingenierie des Protocoles (CFIP'2002), Montreal, Canada*, May 27-30, 2002.

[GOT 03a] GOTZHEIN R., "Consolidating and Applying the SDL-Pattern Approach: A Detailed Case Study", *Information and Software Technology*, vol. 45, 2003, p. 727-741, Elsevier.

[GOT 03b] GOTZHEIN R., KHENDEK F., SCHAIBLE P., "Micro Protocol Design - The SNMP Case Study", SHERRATT E., Ed., *Telecommunications and Beyond - The Broader Applicability of SDL and MSC*, vol. 2599 of *LNCS*, Springer, 2003, p. 61-73.

[GRA 03] GRAMMES R., GOTZHEIN R., MAHR C., SCHAIBLE P., SCHLEIFFER H., "Industrial Application of the SDL Pattern Approach in UMTS Call Processing Development - Experience and Quantitative Assessment", REED R., REED J., Eds., *SDL 2003: System Design*, vol. 2708 of *LNCS*, Springer, 2003, p. 101-115.

[GRA 06] GRAMMES R., "Formal Operations for SDL Language Profiles", GOTZHEIN R., REED R., Eds., *System Analysis and Modeling: Language Profiles*, vol. 4320 of *LNCS*, Springer, 2006, p. 49-63.

[ITU 01] ITU-T, *Message Sequence Charts (MSC)*, ITU-T Recommendation Z.120, International Telecommunications Union, 2001.

[ITU 02] ITU-T, *Specification and Description Language(SDL)*, ITU-T Recommendation Z.100, International Telecommunications Union, August 2002.

[KUH 05] KUHN T., GERALDY A., GOTZHEIN R., ROTHLÄNDER F., "ns+SDL - The Network Simulator for SDL System", PRINZ A., REED R., REED J., Eds., *SDL 2005: Model Driven*, vol. 3530 of *LNCS*, Springer, 2005, p. 103-116.

[KUH 06a] KUHN T., "MacZ - A QoS MAC Layer for Ambient Intelligence Systems", PFEIFER T., Ed., *Advances in Pervasive Computing 2006, Adjunct Proceedings of the 4th International Conference on Pervasive Computing, Dublin, Ireland*, May 7-10, 2006.

[KUH 06b] KUHN T., GOTZHEIN R., WEBEL C., "Model-driven Development with SDL - Process, Tools, and Experiences", *ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS / UML 2006), Genua, Italy*, Oct 1-6, 2006.

[MIL 03] MILLER J., MUKERJI J., *MDA Guide Version 1.0.1*, Object Management Group (OMG), 2003.

[ns2] Information Sciences Institute, University of Southern California, "The Network Simulator ns-2", `www.isi.edu/nsnam/ns`.

[Röß 98] RÖSSLER F., GEPPERT B., SCHAIBLE P., "Re-Engineering of the Internet Stream Protocol ST2+ with Formalized Design Patterns", *Proceedings of the Fifth International Conference on Software Reuse (ICSR5), Victoria, Canada*, 1998.

[SCH 03] SCHAIBLE P., GOTZHEIN R., "Development of Distributed Systems with SDL by Means of Formalized APIs", REED R., REED J., Eds., *SDL 2003: System Design*, vol. 2708 of *LNCS*, Springer, 2003, p. 317-334.

[SHE 03] SHERRATT E., *Telecommunications and beyond: The Broader Applicability of SDL and MSC*, vol. 2599 of *LNCS*, Springer, 2003.

[TAU] Telelogic AB, "Telelogic Tau Generation 1", `www.telelogic.com/products/tau/index.cfm`.

[UML04] Object Management Group (OMG), "Unified Modeling Language 2.0 Infrastructure - Final Adoped Specification", 2004, `www.omg.org/cgi-bin/doc?ptc/2003-09-15`.

[WEB 06] WEBEL C., FLIEGE I., GERALDY A., GOTZHEIN R., KRÄMER M., KUHN T., "Cross-Layer Integration in Ad-Hoc Networks with Enhanced Best-Effort Quality-of-Service Guarantees", *Proc. of the World Telecommunications Congress, Budapest, Hungary*, 2006.

[ZIT 93] ZITTERBART M., STILLER B., TANTAWY A., "A Model for Flexible High-Performance Communication Subsystems", *IEEE Journal on Selected Areas in Communications*, vol. 11, num. 4, 1993, p. 507-518.