Arctis and Ramses: Tool Suites for Rapid Service Engineering

Frank Alexander Kraemer
Norwegian University of Science and Technology (NTNU),
Department of Telematics, kraemer@item.ntnu.no

For our highly automated service engineering approach SPACE, we built the tool suites Arctis and Ramses. Arctis focuses on abstract, reusable service specifications that are composed from UML 2.0 collaborations and activities. It supports the analysis of service specifications by model checking via TLC. A consistent specification can be transformed into UML state machines and components. For their implementation, Ramses contributes code generators to create executable systems.

1 Introduction — The SPACE Approach

As most services involve several participating components, our engineering approach for reactive systems SPACE [8–12] uses collaborative building blocks as reusable specification units to create more comprehensive services through composition. Figure 1 outlines the approach. A developer first consults a library to check if an already existing collaboration block or a combination of several blocks solves a certain task. Missing blocks can also be created from scratch and stored in the library for later reuse. The building blocks are expressed as UML models. The structural aspect, for example the participants and their multiplicity, is expressed by means of UML 2.0 collaborations. For the detailed behavior, we use UML 2.0 activities. They express the local behavior of each of the participants as well as their necessary interactions in a compact and self-contained way using explicit control flows. Coupling points later used for behavioral composition can be expressed by input and output parameter nodes. To define in which sequence these nodes may be invoked by the environment of a building block, we use so-called external state machines (ESMs). They refer with their transitions to the input and output nodes and define as such the externally visible behavior of a collaborative building block.

In a second step, the building blocks are combined to form more comprehensive services by composition. For this composition, we use UML 2.0 collaborations and activities as well. While collaborations provide a good overview of the structural aspect of the composition, i.e., which sub-services are reused and how their collaboration roles are bound, activities express the detailed coupling of their respective behaviors. Each sub-service is represented by a call behavior action referring to the respective activity of the building block. By connecting the individual input and output pins of the call behavior actions, the events occurring in different collaborations can be coupled with each other. The surrounding activity may also contain additional nodes to add further behavior, so that developers have powerful means to express the behavioral composition of the sub-services. Once complete, the service specifications are analyzed using model checking, as we will see later. Note that steps 1 and 2 can be executed iteratively, over several hierarchies, as each composed collaboration can in turn be abstracted by an ESM and used as a building block in more comprehensive specifications.

This paper was presented at the NIK-2007 conference; see *http://www.nik.no/*.

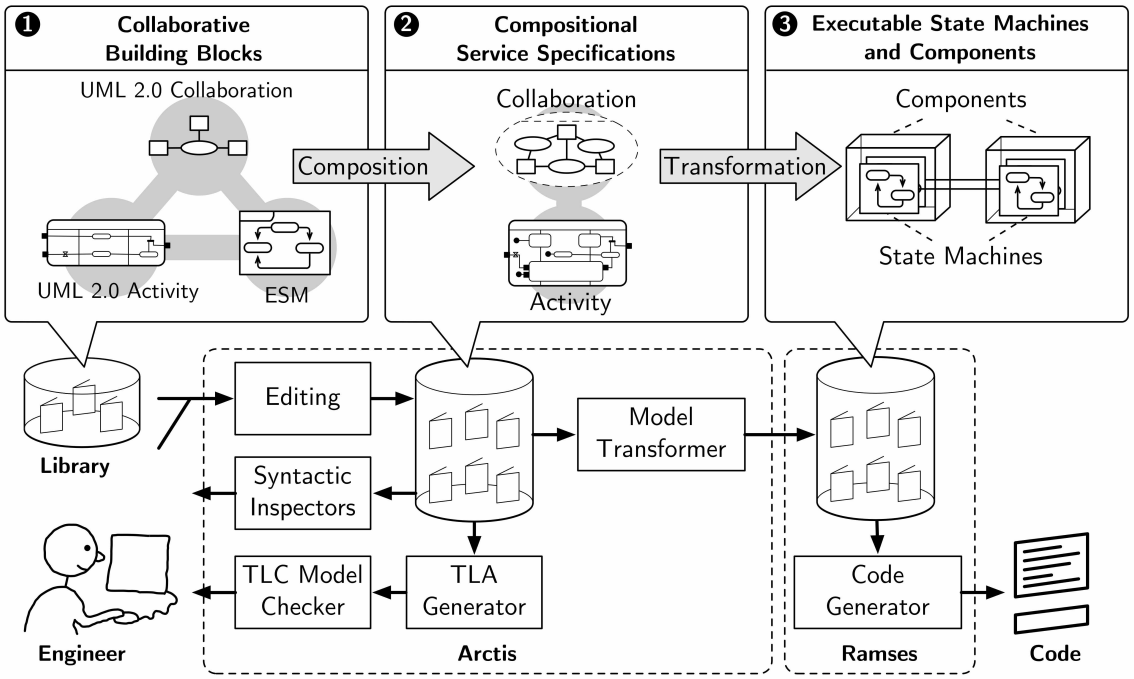

Fig. 1. A coarse sketch of the SPACE engineering approach and its tool support

In a third step, the collaborative service specifications are transformed automatically into executable state machines and components by a model transformation. From these models, executable code can be generated easily in a successive step. So, the only manual work is done on the building blocks and their composition.

To ensure the consistency of the approach, the semantics of its languages, the composition mechanisms as well as the model transformation, we use Herrmann's compositional Temporal Logic of Actions (cTLA, [4]) as formal framework. We have formalized the activities [10] as well as the state machines [12]. The composition of service specifications can be reduced to the mechanism of process couplings in cTLA and the model transformation is a formal refinement step. Practitioners need not understand this formalism and they can focus on SPACE as an approach devoted to support the rapid creation of services. The approach relies on three principles to speed up development:

- **Collaborative Building Blocks.** By focusing on building blocks that are not only components but entire collaborations covering behavior of several components in a self-contained way, services may be composed of sub-services. This facilitates reuse, as sub-services typically serve an isolated task and are more likely to be useful in other applications than entire components.
- **Model Transformations and Code Generation.** By using automated steps from the more abstract service specifications towards the implementation, consistency is ensured between the specifications and coding errors are avoided. As state machines are generated automatically, a difficult and time-consuming manual synthesis is omitted.
- **Formal Analysis of Models.** By analyzing the abstract service specifications both in syntax and via model checking, errors can be found early during the development and on a high abstraction level, where problems can be studied independent from implementation details.

To effectively support these concepts, we have tool support [6] in the form of Arctis for the specification via collaborative building blocks as well as Ramses for the implementation of executable state machines. Both tools are realized as Eclipse plug-ins using the UML 2.0 repository of the UML2 project.

2 Support for Service Specifications in Arctis

To support the construction of building blocks consisting of activities, collaborations and ESMs, Arctis offers special actions and wizards, for example to create the skeleton of an ESM from an activity. In addition, a number of inspections ensures the syntactical consistency of building blocks.

For composite building blocks, where activities with their partitions and call behavior actions must be synchronized with the collaboration, special actions are available to update each of them. For example, Arctis automatically generates a corresponding activity for the behavioral specification of the composition. For each collaboration role, an activity partition is created and each collaboration use is represented by a call behavior action with its pins. This skeleton is then completed manually with activity flows and nodes that model the extra logic to couple the sub-collaborations. A number of inspections are executed on the model to check a consistent syntax. To analyze more advanced properties, we use model checking.

Model Checking. Because of the mapping from activities to cTLA presented in [10], each activity diagram corresponds to a temporal formula. With the tool presented in [16], this formula can be generated in form of TLA$^+$ [13] and used as input for the model checker TLC [18]. If a collaboration is composed from sub-collaborations, we only use the abstract ESMs during model checking. This reduces the state space significantly. The superposition principle of cTLA as well as a formal refinement relationship between an activity and its ESM guarantee that also the complete system behaves correctly.

Model checking requires knowledge that we do not expect from the users of our tools. Therefore, not only the TLA specifications but also the theorems to ensure correct activities are generated automatically, either based on simple assertions in the form of stereotypes attached to the activity, or based on standard assumptions about what makes an activity well-formed. In addition, it is technically possible to project an error trace from TLC which documents erroneous behavior back into the activity diagram as token movements. In this way, the results of model checking can be given to the users, without them needing to know about temporal logic.

Transformation to State Machines. To create executable systems, we implemented an algorithm [11] performing a model transformation from activities to state machines. The basic idea of this algorithm is to cut an activity into its partitions that correspond to different state machines. A token marking corresponds to a control state of a state machine, and each token movement is mapped to a state machine transition. The algorithm explores the state space of the activities by a partial model checking and constructs the necessary state machine transitions. For collaborations that have to be executed in several parallel sessions within one component, the transformation creates session state machines as described in [8].

3 Support for State Machines and Components in Ramses

The models used for the design and execution of services goes back to early computer-controlled telecommunication systems and is based on communicating state machines. With its version 2.0, UML can be used to express such models. As general UML state machines allow behaviors that can be difficult to implement, we defined in [12] a number of constraints on state machines, which may be ensured by syntactic inspections. In addition, state machines can be analyzed using validation algorithms based on the work of Floch [2] and Sanders [14]. These algorithms are currently developed further as part of the SIMS project [15]. Currently, Ramses includes code generators [7, 17] for various versions of the ServiceFrame platform [1], a framework based on the state machine execution mechanisms of JavaFrame [3].

4 Concluding Remarks

The functionality of Arctis will be expanded within the NFR-funded research and development project ISIS (Infrastructure for Integrated Services, [5]). This project incorporates NTNU, HiA, Telenor, Tellu and Ericsson and is working on the development of service applications in the domain of mobile home automation. One aim is to introduce further abstraction levels in Arctis and develop the principle of creating services from reusable collaboration building blocks.

References

1. R. Bræk, K. E. Husa, and G. Melby. *ServiceFrame Whitepaper*. Ericsson NorARC, 2002.
2. J. Floch. *Towards Plug-and-Play Services: Design and Validation using Roles*. PhD thesis, NTNU, 2003.
3. Ø. Haugen and B. Møller-Pedersen. JavaFrame — Framework for Java Enabled Modelling. In Proceedings of Ericsson Conference on Software Engineering, September 2000.
4. P. Herrmann and H. Krumm. A Framework for Modeling Transfer Protocols. *Computer Networks*, 34(2):317–337, 2000.
5. ISIS Project Website. http://www.isisproject.org/, 2007.
6. F. A. Kraemer. The Ramses and Arctis Tools. http://www.item.ntnu.no/∼kraemer/tools.
7. F. A. Kraemer. Rapid Service Development for Service Frame. Master's thesis, University of Stuttgart, 2003.
8. F. A. Kraemer, R. Bræk, and P. Herrmann. Synthesizing Components with Sessions from Collaboration-Oriented Service Specifications. *SDL 2007*, LNCS 4745. Springer–Verlag, 2007.
9. F. A. Kraemer and P. Herrmann. Service Specification by Composition of Collaborations — An Example. In *Proceedings of the 2006 WI-IAT Workshops (IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology)*, pages 129–133, 2006, Hong Kong.
10. F. A. Kraemer and P. Herrmann. Formalizing Collaboration-Oriented Service Specifications using Temporal Logic. In *Networking and Electronic Commerce Research Conference (NAEC)*, 2007.
11. F. A. Kraemer and P. Herrmann. Transforming Collaborative Service Specifications into Efficiently Executable State Machines. In *Procs. of the 6th Int. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)*, volume 7, 2007.
12. F. A. Kraemer, P. Herrmann, and R. Bræk. Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services. In *Proc. of the 8th Int. Symp. on Distributed Objects and Applications (DOA), 2006*, LNCS 4276, pages 1613–1632. Springer–Verlag, 2006.
13. L. Lamport. *Specifying Systems*. Addison-Wesley, 2002.
14. R. T. Sanders. *Collaborations, Semantic Interfaces and Service Goals: a way forward for Service Engineering*. PhD thesis, NTNU, 2007.
15. SIMS Project Website. http://www.ist-sims.org/, 2007.
16. V. Slåtten. Model Checking Collaborative Service Specifications in TLA with TLC. Project Thesis, August 2007. NTNU.
17. A. K. Støyle. Service Engineering Environment for AMIGOS. Master's thesis, NTNU, 2004.
18. Y. Yu, P. Manolios, and L. Lamport. Model Checking TLA$^+$ Specifications. In *Procs. of the 10th IFIP WG 10.5 Advanced Research Working Conf. on Correct Hardware Design and Verification Methods (CHARME'99)*, LNCS 1703, pages 54–66. Springer–Verlag, 1999.