Service Specification by Composition of Collaborations — An Example

Frank Alexander Kraemer Peter Herrmann
Norwegian University of Science and Technology (NTNU), Department of Telematics
N-7491 Trondheim, Norway, {kraemer, herrmann}@item.ntnu.no

Abstract

We outline a specification style for reactive services that focuses on UML 2.0 collaborations and activities as reusable specification building blocks. In contrast to traditional component-based approaches, a collaboration directly describes the interactions between the components as well as the internal behavior necessary for a component to take part in it. To compose services from such reusable collaborations, we use events identified as input and output pins on the activities that are connected together. While our approach is formally settled in temporal logic, in this paper we focus on an example specification from the viewpoint of a service engineer.

1. Introduction

In an accelerated market for mobile communications, with its variety of devices, technologies and operators, the rapid provisioning of new services is an important competitive factor to meet customer demands. A closer look on services reveals that we often find identical or slightly adjusted patterns of functionalities in several services, so that the idea of combining services from existing elements suggests itself. Combining systems from components has been done for decades. The notion of components, however, is orthogonal to that of services, as a service typically requires the coordinated effort of several cooperating components. Therefore, service descriptions are scattered among the component descriptions, and reusing a service would mean to combine parts of various component descriptions, which implies a lot of manual work and is error-prone.

Instead, by specifying the interactions of the participating components as well as their required local behavior in form of so-called collaborations, we can describe services in a self-contained way. Collaborations express patterns of behavior between several participants and may either be elementary or composed from other ones. Thus, they enable a form of reuse that goes beyond that of reusing (physically concentrated) components; existing collaborations involving several components may be directly reused as well, so that services can be created by combining them from sub-services. Some service examples were presented in [5, 10, 13, 14], and experience showed that focusing on collaborations results in service specifications that are quite intuitive to understand.

The challenge of service engineering is the correct composition of service patterns specified by collaborations. While we use the compositional Temporal Logic of Actions (cTLA, [8]) to describe compositions of collaborations in form of process compositions, in this paper we focus on UML 2.0 collaborations and activities as a notation convenient for service engineers. As an example, we utilize an access control system (ACS), which first appeared in [3] and was later adapted in [4]. The system is used to restrict the opening of a door to a group of trusted people which have to identify themselves with an identity card and a personal identification number. This identification data (referred to as *pid*) can be provided at a panel next to the door. The panel and the door opening mechanisms are handled by a local control station which is also installed close to the door. The local station is connected with a central station calling both an authentication and an authorization service which check the pid by accessing corresponding databases. Figure 1 depicts a UML 2.0 collaboration modeling the cooperations between the various components by means of collaboration uses. The local station accesses the panel and the door opener according to the collaboration uses *p* and *d* while its interaction with the central node is specified by *t*. The central station verifies the personal data by calling the authentication and authorization services modeled by the collaboration uses *a1* and *a2*. Their corresponding servers obtain the needed data from databases as described by the collaboration uses *r1* and *r2*.

While the UML collaborations describe the structural aspects of the composed services (i.e., the relation between components and collaboration roles), we specify the behavioral aspect by means of UML 2.0 activities. In particular, we use activities for both elementary collaborations and the systems combined from collaborations. For example, in Fig. 3, we model the behavior of the access control system.

Proceedings of the 2006 IEEE/WIC/ACM International Conference on
Web Intelligence and Intelligent Agent Technology (WI-IAT 2006 Workshops)(WI-IATW'06)
0-7695-2749-3/06 $20.00 © 2006 IEEE

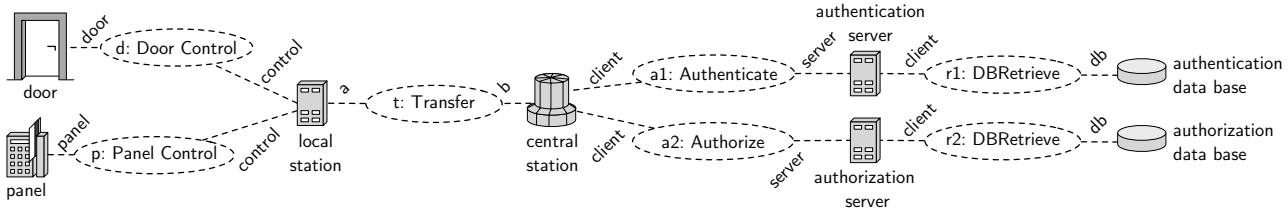

Figure 1. Access control system composed of collaborations

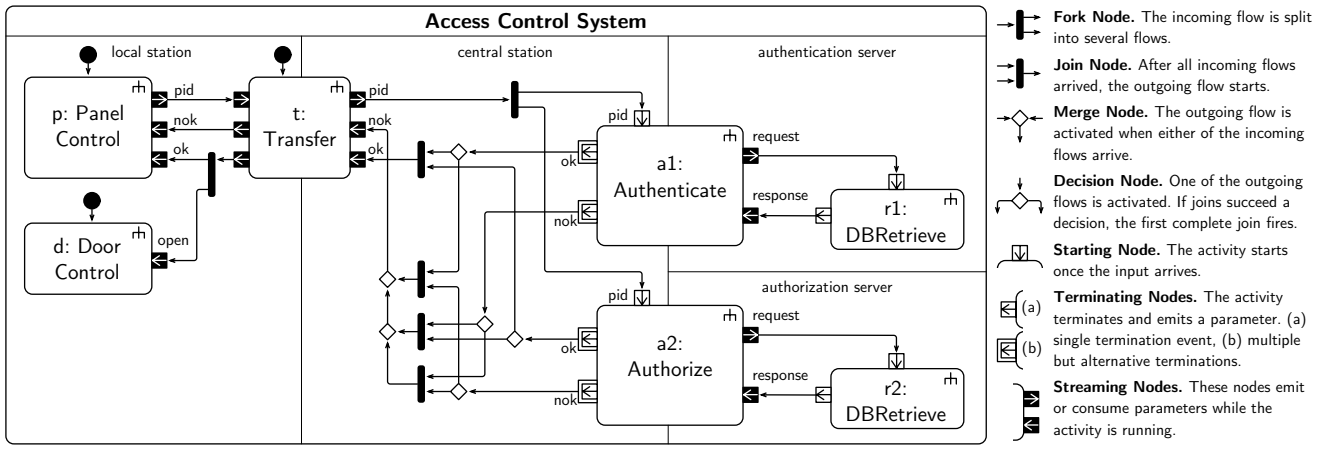

Figure 2. System activity to couple the collaborations of the access control system

Hereby, each collaboration use of Fig. 1 is represented by a call behavior action referring to the activity describing the behavior of the corresponding collaboration. In addition, the activity contains control nodes and flows to couple the different collaborations with each other by connecting their input and output pins. The details of Fig. 2 will be explained in Sect. 3.

The structure of collaborations as well as the way to couple them facilitates the reuse of activities. For example, both the collaboration uses *r1* and *r2* are identical and can be instantiated from a single collaboration type. Moreover, the collaboration uses *a1* and *a2* are very similar and can be based on the same UML template. Thus, systems of a specific domain can often be composed of reoccurring building blocks that are selected from existing libraries. The composition of these building blocks is specified by pairs of a UML collaboration and a UML activity as exemplified in Fig. 1 and Fig. 2. These abstract diagrams give (in our opinion) a good overview of the system and are rather intuitive.

Another advantage of collaboration-oriented modeling is the fact that it supports the convergence of information and communication technology modeling concepts (cf. [2]). In particular, we can encapsulate telecommunication-specific properties (e.g., complex synchronization aspects of peer-to-peer communication) within collaborations. Thus, developers familiar with the concepts of the computing domain but not with the telecom domain should have less trouble to create sound distributed systems. One can further understand collaborations as a novel way to describe component interfaces. In contrast to behavioral interfaces (cf. [1]), they may describe not only behavior guaranteed by the component itself but also behavior of the component environment which is helpful to use the component in a suitable way.

In the following, we will exemplify our approach by means of the ACS example. In Sect. 2 we introduce the collaborations of this system in detail while Sect. 3 is devoted to describe the composition of the collaborations to an overall system model.

2. Collaborations

To validate the personal identifiers (pid) provided by a user who requests access through the door, the central node participates in the collaboration *Authenticate* together with the authentication server. This is specified by the collaboration use *a1:Authenticate* in Fig. 1, where the central node plays the role *client* and the authentication plays the role *server*. (The role bindings are depicted as labels in a 45° angle). The behavior of the collaboration is described by the

Proceedings of the 2006 IEEE/WIC/ACM International Conference on
Web Intelligence and Intelligent Agent Technology (WI-IAT 2006 Workshops)(WI-IATW'06)
0-7695-2749-3/06 $20.00 © 2006 IEEE

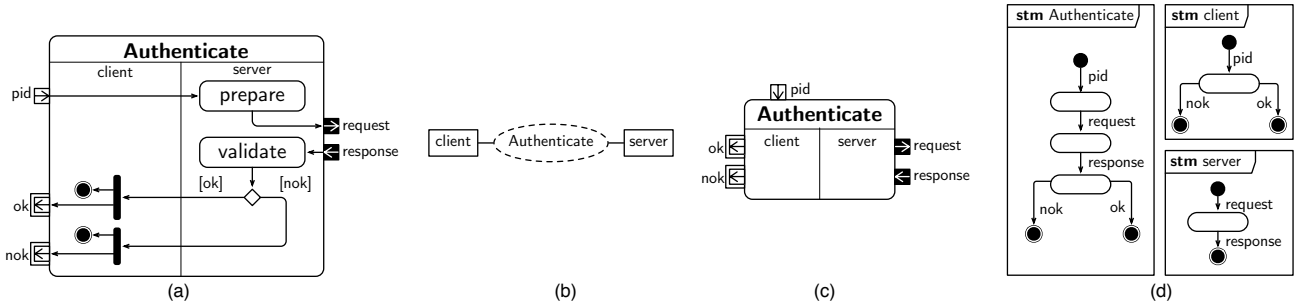

Figure 3. Internal and external specifications for the authentication

UML 2.0 activity in Fig. 3 (a)[1]. The activity is divided in two partitions, one for each collaboration role (*client* and *server*). It is started on the client side, when a pid is provided as parameter at the input pin. The pid is then directly sent to the server, where it is converted into a database request in the call operation action *prepare*. Thereafter it is the task of the collaboration between the server and the database to provide the stored user information. To get that information, the request leaves the activity *Authenticate* and the server waits for the reception of the response. This is modeled with the input and output pins *request* and *response*. (As the parameters may pass the border of the activity while it stays active, these are so-called streaming parameters, denoted by the filled square.) Depending on the validity of the pid, the server may decide to report *ok* or *nok* to the client. This result is forwarded to the corresponding output pin, and the activity is terminated by an activity final node (◉). (Since *ok* and *nok* are alternative output pins, they must belong to different parameter sets, indicated by an additional box around the output pin.)

As mentioned in the introduction, the authorization, where the central station asks an authorization server whether the identified person has access to the requested area, behaves similarly. We can therefore create a common collaboration template and put it into a library of reusable service building blocks. To provide an easier access, we supply the collaboration with a more abstract description that hides the details of its internal realization. This abstract description consists of the declaration of the UML collaboration (Fig. 3 b) as well as the description of the activity with its externally visible pins (Fig. 3 c). Each pin corresponds to an event that is visible to the participants. The assignment of a pin to one of the activity partitions shows which collaboration role can observe or cause an event. In addition, we define the external behavior (i.e., the order of events passing the pins) by means of state machines. For example, state machine *Authenticate* in Fig. 3 (d) explains that the start of the collaboration by providing the pid leads to a request. The name of each transition corresponds to the name of a pin of the activity. The state machines may cover all events (as in *stm Authenticate*) or only those visible to the individual collaboration roles, like for the state machines *stm client* and *stm server*.

Depending on the response from the central node, the local node may open the door, which is expressed in the collaboration *Door Control* in Fig. 4. We consider the door to be part of the environment, and mark its activity partition therefore as «external». The door mechanism simply receives signals from the controller that activates or deactivates a magnet opening the lock. On the controller side, a timer automatically triggers the sending of the signal to close the lock after a while. Therefore, the local station participating in this collaboration only has to trigger the opening of the door. This collaboration is a simple example of including useful environment behavior in an interface description. Namely, the collaboration can, for instance, be an interface description provided by the producer of the door control system, which suggests a useful function to the controller of the door (i.e., the timeout mechanism).

3. Coupling and Composition

While the collaboration in Fig. 1 covers the structural aspect of the composed system, we use the activity in Fig. 2 to describe how the events of the individual collaborations be-

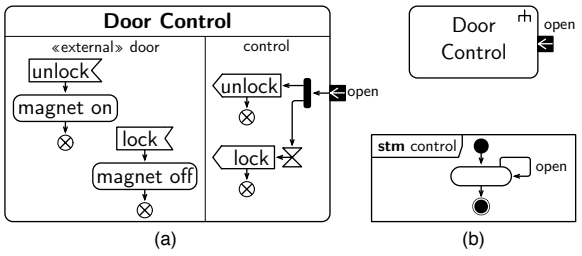

Figure 4. Description for the door control

[1]Note that a UML 2.0 collaboration, besides being a *structured classifier* with parts and connectors, is also a *behaviored classifier* that can describe its behavior for example by means of an UML 2.0 activity [12].

Proceedings of the 2006 IEEE/WIC/ACM International Conference on
Web Intelligence and Intelligent Agent Technology (WI-IAT 2006 Workshops)(WI-IATW'06)
0-7695-2749-3/06 $20.00 © 2006 IEEE

tween the system components are coupled with each other, so that the desired overall system behavior is obtained. Therefore, the activity contains a separate call behavior action for each collaboration use of the system. As outlined in Sect. 1, each of these call behavior actions refers to the activity describing the behavior of the corresponding collaboration. The initial nodes (●) mark the collaborations (or activities, resp.) that are started together with the entire system. When a user requests access via the panel, *p: Panel Control* will emit a pid that is transfered to the central station via collaboration *t: Transfer*. Once arrived at the central station, both the authentication and authorization collaborations are started in parallel which is expressed by the fork node in the partition of the central station. According to the activity in Fig. 3 (a), *Authenticate* makes a database request, modeled by collaboration *r1: DBRetrieve* and terminates with one of the alternative results *ok* or *nok*. The collaboration for the authorization behaves similarly. The coupling of the results of the authentication and authorization is modeled by the decision, join and merge nodes in the partition of the central station. Obviously, an *ok* may only be sent to the local station if both the authentication and the authorization terminated with an *ok*. In all other cases, the response is *nok*. If the local station receives *ok*, it displays this on the panel and simultaneously opens the door via collaboration *d: Door Control*, which is modeled by the fork in the local station. Otherwise, the panel indicates the denied access to the user and the door remains closed.

The activity in Fig. 2 and the collaboration in Fig. 1 show how reusable specification building blocks in form of collaborations can be composed together. Of course, there are rules that have to be obeyed when the activity diagrams, state machines and collaborations are created:

- The activity diagram must be kept consistent with the collaboration it describes. In particular, for each collaboration use there must be a call behavior action pointing to the corresponding activity, and each activity partition has to correspond to a collaboration role. Moreover, the role bindings of the collaborations must be consistent with the assignments of the pins to the activity partitions.

- In an activity, there must always be exactly one triggering event (like an output or termination) in a set of connected pins. In particular, two output pins may not be connected directly. Instead, they may, for example, be connected via a merge node (◇), describing that any of them can trigger a third event.

- If an input pin declares a parameter, it must be provided by a connected object flow, so that data types are consistent.

These rules can easily be checked by inspecting the UML model syntactically. Besides them, there are properties that may require model checking. For example, if the behavior of an activity corresponds to the interface description given with the state machine, or if a composition of collaboration can actually terminate. We outline our way to check these descriptions within the explanation of the tool suite below.

In the example in Fig. 2, we use only edges linking nodes within a particular partition guaranteeing an easy transformation of the collaboration-oriented model into a component-oriented model. Nevertheless, one can also use edges crossing partition boundaries. These edges are refined to signal transfers between the components realizing the linked partitions.

4. Concluding Remarks

The collaboration-oriented specification style introduced above is a step of a more comprehensive service engineering approach depicted in Fig. 5. Services are specified by creating, composing, and refining collaborations as described in this paper. Whenever possible, one may reuse existing collaboration building blocks gathered in a library. Syntactic inspections of the service specification give feedback to the user, for example, whether the activity diagrams obey the rules and constraints as outlined above. Once a service specification is consistent, it may, in turn, be added to the library for later reuse. For executing a service, the specification may be transformed into UML 2.0 components and state machines, which are a suitable input for our code generators producing implementations for various service execution platforms [10].

To ensure the correctness of service compositions and the model transformations, we use the linear-time temporal logic cTLA [8] as an underlying formalism. As UML activities have a *"Petri-net like semantics"* [12], they describe a state transition system that can easily be expressed by cTLA processes [6]. The coupling of collaborations by means of input and output pins of the corresponding activities, introduced here, can be directly mapped to the composition of processes by joint actions in cTLA. Besides this kind of coupling, cTLA also facilitates the so-called constraint-oriented specification style [15] where certain constraints of a collaboration can be specified separately. This coupling principle can also be used in UML activities facilitating reuse further. It enables several UML actions of different collaborations to be joined with each other so that they are carried out synchronously. cTLA also provides the concept of specification frameworks (cf. [7, 8, 9]) which are a complement to the collaboration libraries discussed above.

This double appearance of our service specifications both as UML models and as temporal logic formulas enables us to analyze them formally. A transformation into

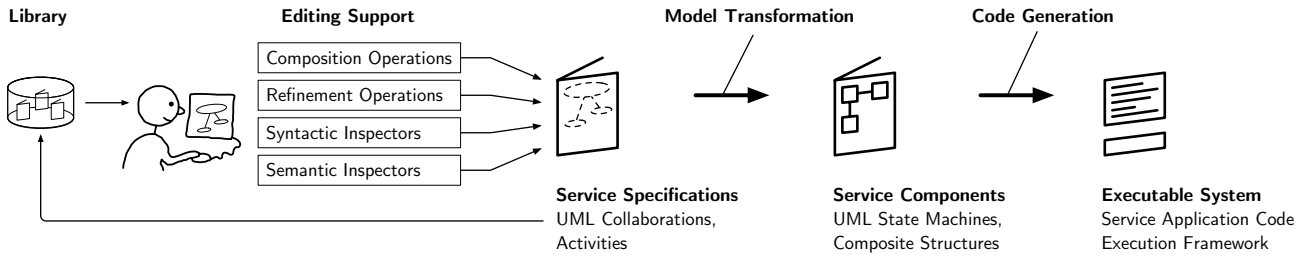

Figure 5. Approach for service engineering

TLA$^+$[11] makes service specifications a possible input for the model checker TLC [16], examining them for semantic properties. In particular, one must prove that an internal collaboration fulfills the properties of the external description, which corresponds to a refinement proof in cTLA.

Logical reasoning is also needed to verify the step that carries a service specification based on collaborations and activities over to the executable form based on UML components and state machines. While we implemented this step in form of a UML model transformation, we use cTLA refinement proofs to verify its correctness. For this reason, we described the cTLA style cTLA/e [10] that aligns the semantics of UML 2.0 state machines with the mechanisms of our execution platforms. Thus, we formally established the target of the refinement process. This model transformation will be part of our tool suite ARCTIS for the analysis, refinement, composition and transformation of interactive services, supporting the development process all the way from the early specification until executable systems.

References

[1] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins. Making Components Contract Aware. *IEEE Computer*, 32(7):38–45, July 1999.

[2] R. Bræk and J. Floch. ICT Convergence: Modeling Issues. In D. Amyot and A. W. Williams, editors, *SAM'04 - Fourth SDL and MSC Workshop*, volume 3319 of *Lecture Notes in Computer Science*, pages 237–256. Springer, 2004.

[3] R. Bræk and Ø. Haugen. *Engineering Real Time Systems: An Object-Oriented Methodology Using SDL*. The BCS Practitioner Series. Prentice Hall International, 1993.

[4] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.

[5] H. N. Castejón and R. Bræk. A Collaboration-based Approach to Service Specification and Detection of Implied Scenarios. *ICSE's 5th Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM'06)*, 2006.

[6] G. Graw and P. Herrmann. Transformation and Verification of Executable UML Models. *Electronic Notes on Theoretical Computer Science, Elsevier Science*, 101:3–24, 2004.

[7] P. Herrmann. Formal Security Policy Verification of Distributed Component-Structured Software. In H. König, M. Heiner, and A. Wolisz, editors, *Proceedings of the 23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'2003), Berlin, Germany*, volume 2767 of *Lecture Notes in Computer Science*, pages 257–272. Springer-Verlag, September 2003.

[8] P. Herrmann and H. Krumm. A Framework for Modeling Transfer Protocols. *Computer Networks*, 34(2):317–337, 2000.

[9] P. Herrmann and H. Krumm. A Framework for the Hazard Analysis of Chemical Plants. In *Proceedings of the 11th IEEE International Symposium on Computer-Aided Control System Design (CACSD'2000)*, pages 35–41, Anchorage, 2000. IEEE CSS, Omnipress.

[10] F. A. Kraemer, P. Herrmann, and R. Bræk. Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services. In R. Meersmann and Z. Tari, editors, *Proceedings of the 8th International Symposium on Distributed Objects and Applications (DOA), 2006, Montpellier, France*, volume 4276 of *Lecture Notes in Computer Science*, pages 1613–1632. Springer–Verlag Heidelberg.

[11] L. Lamport. *Specifying Systems*. Addison-Wesley, 2002.

[12] Object Management Group. Unified Modeling Language: Superstructure, April 2006.

[13] J. E. Y. Rossebø and R. Bræk. Towards a Framework of Authentication and Authorization Patterns for Ensuring Availability in Service Composition. In *Proceedings of the 1st International Conference on Availability, Reliability and Security (ARES'06)*, pages 206–215. IEEE Computer Society Press, 2006.

[14] R. T. Sanders, H. N. Castejón, F. A. Kraemer, and R. Bræk. Using UML 2.0 Collaborations for Compositional Service Specification. In *ACM / IEEE 8th International Conference on Model Driven Engineering Languages and Systems*, 2005.

[15] C. A. Vissers, G. Scollo, M. van Sinderen, and H. Brinksma. Specification Styles in Distributed System Design and Verification. *Theoretical Computer Science*, 89:179–206, 1991.

[16] Y. Yu, P. Manolios, and L. Lamport. Model Checking TLA+ Specifications. In L. Pierre and T. Kropf, editors, *Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'99)*, volume 1703 of *Lecture Notes in Computer Science*, pages 54–66. Springer-Verlag, 1999.

Proceedings of the 2006 IEEE/WIC/ACM International Conference on
Web Intelligence and Intelligent Agent Technology (WI-IAT 2006 Workshops)(WI-IATW'06)
0-7695-2749-3/06 $20.00 © 2006 IEEE