

Methodology for Model-based Dynamic Composition of Services

Judith E. Y. Rossebø^{1,2} Ragnhild Kobro Runde³

¹Department of Telematics, NTNU, Norway

²Telenor GBDR, Norway

³Department of Informatics, University of Oslo, Norway

Abstract

In order for service providers to be able to proactively keep up with the rapidly changing and converging telecommunications environment, there is a need to be able to design and develop services so that they may be deployed and adapted dynamically. This paper presents a general methodology for model-driven, policy-enabled dynamic composition of services. The methodology addresses the problem of how to model dynamic composition of services, and uses a three layered approach to service specification, design and deployment in order to realize dynamic service execution in the runtime system. The method is explained through its use in a tele-consultation case.

1 Introduction

This paper presents a general methodology for model-driven development of distributed services composed of several service parts. A service is an identified functionality (i.e. behaviour) involving a collaboration among entities in order to achieve some desired goals/effects for end users or other entities. One important aspect of services is that they normally are partial functionalities that cross-cut the component structure so that a service may involve several collaborating components while each component may participate in several services.

Service composition addresses the means of composing the partial behaviours of collaborating components in order to achieve the overall service. Service composition allows for incremental service development, for which components may be statically or dynamically combined, and it allows for reuse of components.

In this paper, we address the problem of how to model dynamic composition of services. We model a service as a collaboration among roles with associated behaviour, using UML 2.x [1]. The basic building blocks for composed services are elementary service collaborations between two entities collaborating on an interface to provide a service or a service feature [2]. The visible interface behaviour associated with an elementary collaboration is modelled using UML 2.x interactions.

In [3], we introduced a policy-driven approach to service modelling, design, and implementation and we presented the concept of a composition policy and a policy

This paper was presented at the NIK-2009 conference; see <http://www.nik.no/>.

enforcement state machine (PESM) for modelling the choreography of service collaborations. We elaborated on the specification of the PESH diagram and composition policies in [4]. Essentially, the global PESH is similar to a UML 2.x activity diagram with collaboration uses as nodes. The global PESH diagrams used in our approach are similar to the choreographies inspired by UML 2.x activities presented in [5] and further elaborated in [6]. However, those choreographies describe static/non-dynamic composition of collaborations, for which the choreography is hardcoded.

With our approach, dynamic coordination of the elementary collaborations and the conditions for choosing between alternative collaborations, are governed by composition policies. Using policies for this purpose provides flexibility, in that the exact set of elementary collaborations involved in the composed service, and the ordering of these is determined first when the service is actually executed in the runtime system. Separating the specification of the policy rules from the collaborations, also allows for changes to be made at runtime by changing the policy rules without changing the service parts themselves.

An approach to adaptive service composition with many similarities to our approach is presented in [7]. Whereas they use a condition-action rule-based decision process, our policies are based on the event-condition-action paradigm. The decision system architecture is similar to our policy-based architecture, with a distributed management system. However, in [7], the management is modelled more tightly integrated with the services than in our approach, where the policy management is specified separately from the individual services.

The methodology presented in this paper explains how to use UML 2.x, PESHs and composition policies to model, design and deploy dynamic composition of services. We exemplify the methodology using a tele-consultation service.

The rest of the paper is organized as follows: In Sect. 2 we provide an overview of the policy-based approach to dynamic composition and in Sect. 3 we introduce the tele-consultation service used in the remainder of this paper. The methodology is presented in Sects. 4–6, exemplified through the tele-consultation service.

2 Overview of the Policy-Based Approach

A graphical overview of the model-driven, policy-based approach to dynamic service composition is provided in Figure 1. At the *service model layer*, services are modelled structurally using UML 2.x collaboration diagrams [1], and policy rules stored in a table are used for governing dynamic choices among collaborations, i.e. dynamic composition. The policy enforcement state machine (PESH) diagram graphically depicts the choreography of the collaboration uses involved in the composed service. While the service model layer provides a description of the service as a whole, the *design model layer* describes the service behaviour as it is distributed to each separate component. The individual role behaviours are given by UML 2.x state machines, and the coordination of these is specified by local policy rules and local PESH diagrams, derived by transformations from the global models and rules at the service model layer. From the design models, implementations can be generated, which in combination with a policy-management system in the runtime system, enable the actual runtime *realization* of the dynamic composition. As shown in Fig. 1, the policy management is a separate management layer in the deployed system. Further details of this layer may be found in [8].

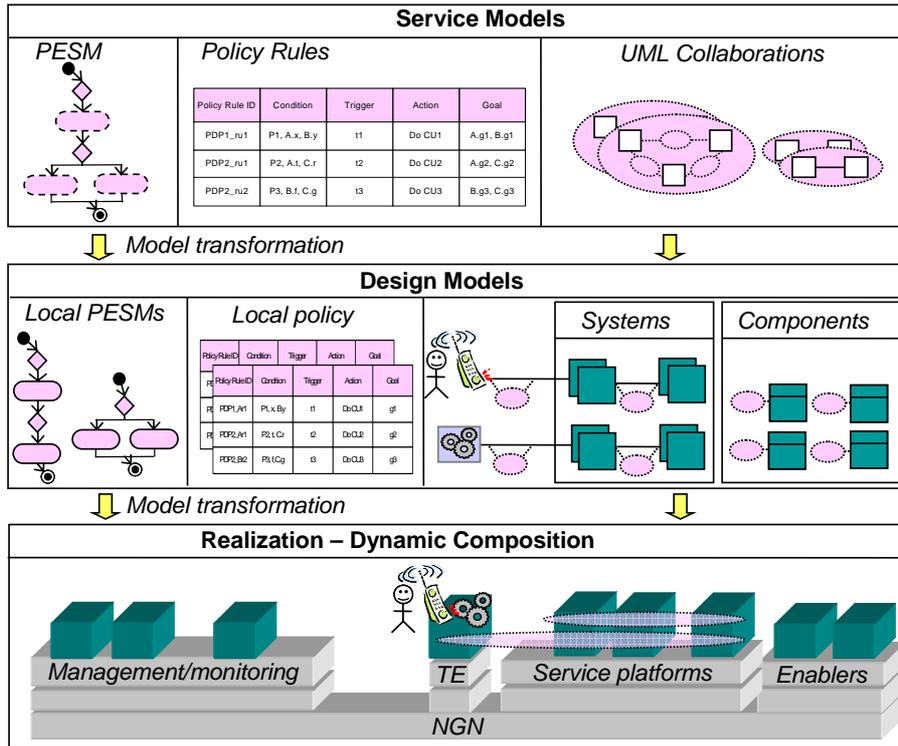


Figure 1: Overview of the policy-enabled approach to service composition

3 Description of the Tele-Consultation Service

The tele-consultation system is designed to allow chronically ill patients to have intermediate consultations from their homes, so as to reduce repeated travel to the doctor’s office/hospital for routine consultations when routine tests may equally well be performed remotely. Equipment is provided in the home for carrying out tests. This equipment collects data from e.g., medical sensors and this information is transmitted/communicated to the doctor.

In order to request a consultation with a doctor, the patient needs to register to an electronic tele-consultation service desk, which then sets up a remote consultation with the patient’s doctor, depending on availability. The patient may register by communicating directly with the electronic tele-consultation service desk, or if the patient is in need of special assistance to register, a human receptionist may be contacted. The receptionist then gets the registration information from the patient, and provides this information to the electronic tele-consultation service desk.

4 Methodology: The Service Model Layer

The methodology follows the three layered approach from Sect. 2. In this section we address the steps involved at the service model layer.

Step 1: Specify the service (or family of services)

Use a UML 2.x collaboration overview diagram to describe:

1. the distributed parts involved in the composed service, given as one composite role for each part.
2. the set of collaboration uses involved in the composed service. Each collaboration use is specified in one of the following ways:

- (a) either, as an elementary collaboration, i.e. a collaboration between exactly two elementary roles,
 - (b) or, as a composed (sub-)service described recursively as a UML 2.x collaboration overview diagram in the same manner as described here.
3. the binding of the (elementary or composite) roles in the collaboration uses to the composite roles from step 1.

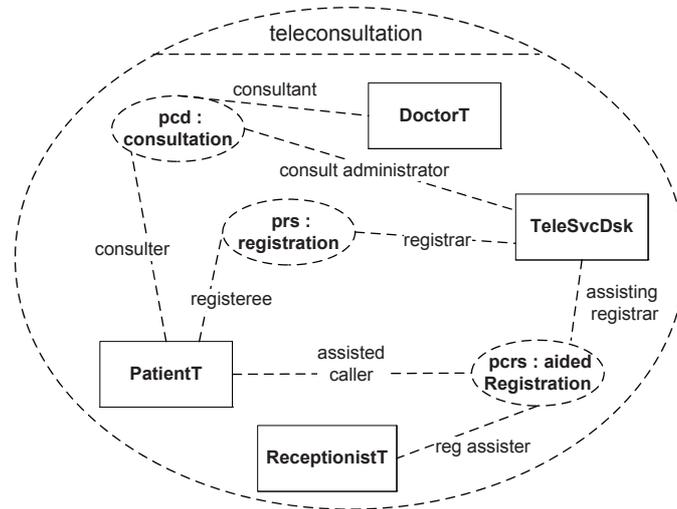


Figure 2: Collaboration overview diagram for the tele-consultation case

The resulting diagram then provides a means of statically and structurally representing the binding of collaboration uses involved in the composed service to the actual service parts. Any of the collaboration uses involved in the composed service may be selected from a repository of re-usable service specifications.

Step 1 applied to the tele-consultation service The UML 2.x collaboration overview diagram produced for the tele-consultation service is shown in Fig. 2. In order to arrive at this collaboration overview, the following steps were taken:

1. The distributed parts involved in the composed service are `PatientT`, `ReceptionistT`, `DoctorT` and `TeleSvcDsk`, given as composite roles in the collaboration overview diagram.
2. The tele-consultation service is composed of three collaboration uses: one for `consultation`, one for `registration` and one for `aided registration`. As will be specified as part of step 3, only one of `registration` and `aided Registration` will be performed in a single run of the service.
 - (a) Although several elementary collaborations are used in the tele-consultation service, we only show one of them, the `call` collaboration, due to lack of space. This elementary collaboration, which is used in the `aided Registration` and `consultation` collaborations, models a two way communication between two parties such as a telephone call and is shown in Fig. 3.
 - (b) There are two composite sub-services used in the tele-consultation service: `aided Registration` and `consultation`. The collaboration diagrams for these are shown in Figs. 4 and 5. In addition to using the elementary `call`

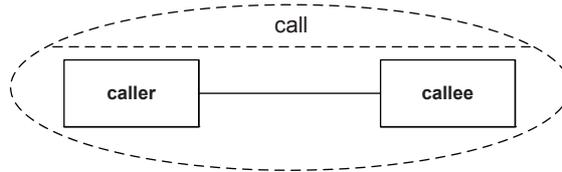


Figure 3: The call collaboration

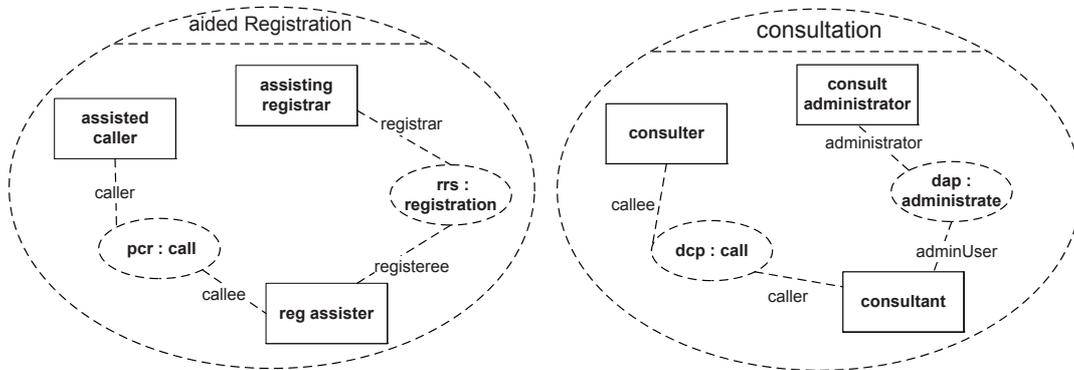


Figure 4: The aided Registration collaboration Figure 5: The consultation collaboration

collaboration, aided Registration also uses the elementary registration collaboration and consultation uses the elementary administrate collaboration.

Intuitively, the aided Registration collaboration is invoked when a patient requires special assistance to register. The exact coordination between the two parts of aided Registration will be described as part of step 3.

3. The binding of the (elementary or composite) roles in the collaboration uses to the composite roles, PatientT, ReceptionistT, DoctorT and TeleSvcDsk, is shown in Fig. 2. For example, for the collaboration use pcrs of the aidedRegistration collaboration, the three composite roles assisted caller, assisting registrar and reg assister are bound to the PatientT, TeleSvcDsk and ReceptionistT, respectively.

Step 2: Specify the behaviour of each elementary collaboration

For each of the elementary collaborations used in the UML 2.x service specification in step 1, if collaboration is not already specified and stored in the repository of re-usable service specifications, specify the visible interface behaviour as a UML 2.x sequence diagram. For ensuring that the behaviour associated with the elementary collaborations can execute correctly during dynamic service execution in the runtime system, the visible interface behaviour between the two elementary roles should be validated with respect to safety and liveness properties. This may be done using static analysis, e.g., by checking the type compatibility of signals exchanged, or using dynamic analysis, that is, taking into consideration the order of events occurring during dynamic execution [9,10,2].

Step 2 applied to the tele-consultation service We illustrate this step by specifying the UML 2.x sequence diagram for the call collaboration from step 1. This sequence diagram is shown in Fig. 6. Here, we use the STAIRS [11,12]

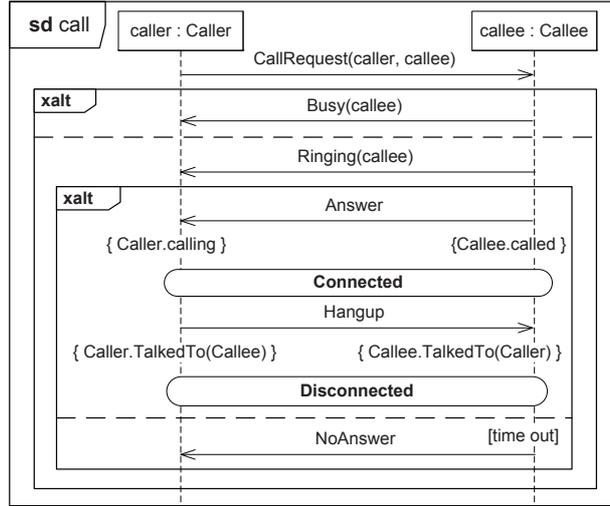


Figure 6: UML 2.x sequence diagram for the `call` collaboration

operator `xalt` to express that each of the alternatives are mandatory in the final implementation. We also use UML state invariants declared on the lifelines to express role goal assertions for each of the roles, e.g., `Caller.TalkedTo(Callee)` and `Callee.TalkedTo(Caller)`, and UML continuation labels, e.g., `Connected`, to express collaboration goal assertions as in [2].

For examples demonstrating validation of the visible interface behaviour between the two elementary roles in the call collaboration with respect to safety and liveness properties, we refer to [2] which uses a very similar call specification to demonstrate the validation techniques.

Step 3: Specify the global PESM diagram

Use a global PESM diagram together with a global composition policy to describe the choreography of the behaviour associated with the UML 2.x collaboration overview diagram from step 1 using

1. a global PESM diagram for describing the ordering of the collaboration uses.
2. a set of global policy rules for each policy decision point (PDP) in the global PESM diagram, using the precise syntax given in [4]. Together, these policy rules constitute the composition policy for the composed service.

Step 3 applied to the tele-consultation service In step 1, we created a service composed of three (sub-)services for which the behaviour needs to be specified here. First, we consider the `aided Registration` collaboration from Fig. 4. The choreography of the associated behaviour is described by the global PESM diagram given in Fig. 7. For convenience of the reader, each policy rule is shown as a note annotated to the appropriate branche of the PESM diagram. In the diagram, we use a simplified notation for the collaboration uses, where e.g. `pcr(caller -> assisted caller, callee -> reg assister):call` stands for the collaboration use `pcr` of the collaboration `call`, where the elementary roles `caller` and `callee` are bound to the composite roles `assisted caller` and `reg assister`, respectively.

During `aided Registration`, the receptionist obtains the registration information from the patient during the call, and provides this information to the electronic tele-consultation service desk. As such, these collaborations are not synchronized, and

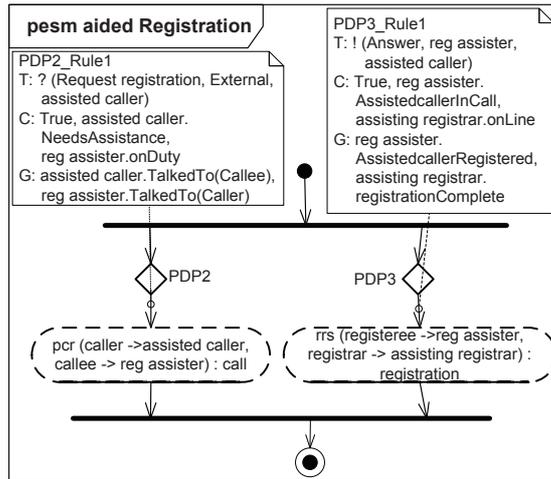


Figure 7: Global PESM diagram for the aided Registration collaboration

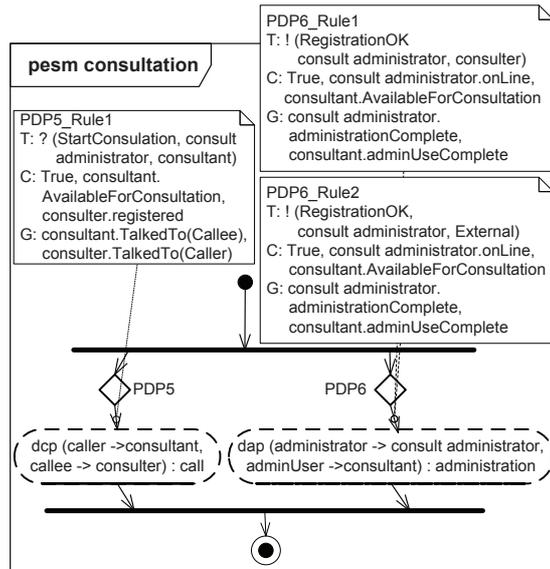


Figure 8: Global PESM diagram for the consultation collaboration

horizontal coordination between the collaboration uses is not shown in this view, as the receptionist is responsible for translating information received from the patient to the appropriate form required for entering into the electronic service desk. This is why the two collaboration uses for `call` and `registration` are shown in parallel in Fig. 7 (using the UML 2.x notation where a heavy bar represents a fork or a join). Evaluation of `PDP2_Rule1` is triggered when the `assisted caller`, bound to the `PatientT` composite role, receives a request for registration. If the given condition evaluates to true, the `call` collaboration should be performed. The goal is that afterwards, both the patient and the receptionist should have reached a status of having talked to each other. Similarly, evaluation of `PDP3_Rule1` is triggered when the `Answer` message of the `call` collaboration (as described in Fig. 6) is sent from the `reg assister` to the `assisted caller`, bound to the `ReceptionistT` and `PatientT` composite roles, respectively.

Next, we describe the behaviour of the `consultation` collaboration, using a global PESM diagram and composition policies as shown in Fig. 8. During `consultation`, the doctor is in a call with the patient. At the same time, the doctor is interacting with the electronic tele-consultation service desk in order to obtain patient data as well as record information in the patient journal.

The composition policy for `consultation` gives that the `administration` collaboration can be performed when either one of the two rules `PDP6_Rule1` and `PDP6_Rule2` applies. As part of the behaviour of the `administration` collaboration, the `consult administrator` role, which is bound to the `TelesvcDsk` composite role, then triggers the evaluation of policy rule `PDP5_Rule1` by sending the `StartConsultation` message to the `consultant` role, which is bound to the `DoctorT` composite role. The `administration` collaboration runs in parallel to the `call` as the doctor will need to look up information in the patient journal and record patient status and test results during the entire consultation by communicating with the electronic tele-consultation service desk. The `call` collaboration can be performed when the policy rule `PDP5_Rule1` applies, that is, when the trigger has been received and the condition part of the policy rule is true.

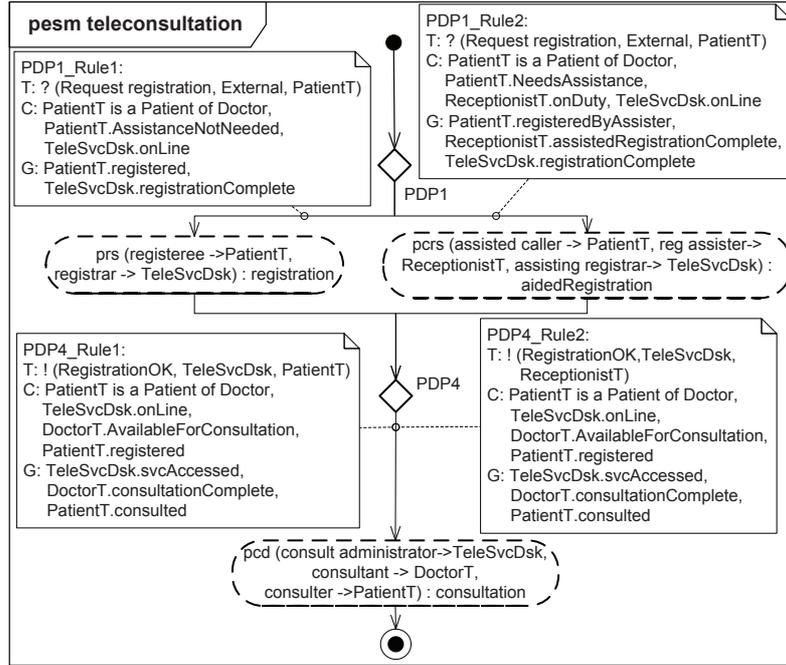


Figure 9: Global PESH diagram for the tele-consultation service

Finally, the choreography of the behaviour associated with the `teleconsultation` collaboration in Fig. 2 is described by the global PESH diagram given in Fig. 9. In this case, the two policy rules given at PDP1 govern the choice between `registration` and `aidedRegistration`. The actual execution of the tele-consultation service will follow one of the choices depicted by the PESH, and the choice is made dynamically each time. For presentation purposes, the PESH diagram only describes successful execution of the service. Unsuccessful or exceptional executions (e.g. if the patient does not belong to the doctor) may be described using additional policy rules and possibly involving other collaboration uses. The example given here shows a choice between two possible dynamic executions of the service. Other examples, demonstrating that the methodology can be used to change the service by adding other collaboration uses and policy rules providing different alternatives for service parts, to be composed dynamically depending on policy, may be found in [8].

5 Methodology: The Design Model Layer

At the design model layer, the overall service behaviour described at the service model layer is broken down for each composite role separately. In this section, we address the methodological steps performed in order to achieve this.

Step 4: Create elementary role state machines

For each elementary collaboration not taken from the repository but specified in step 2, create one elementary role state machine for each of the two roles involved in the collaboration. The transformation from sequence diagrams to elementary role state machines may be performed manually or automatically as described in [13,14].

Step 4 applied to the tele-consultation service Sect. 4 specifies six elementary roles. As examples, Fig. 10 shows the elementary role state machines for the `caller` and `callee` roles involved in the `call` collaboration.

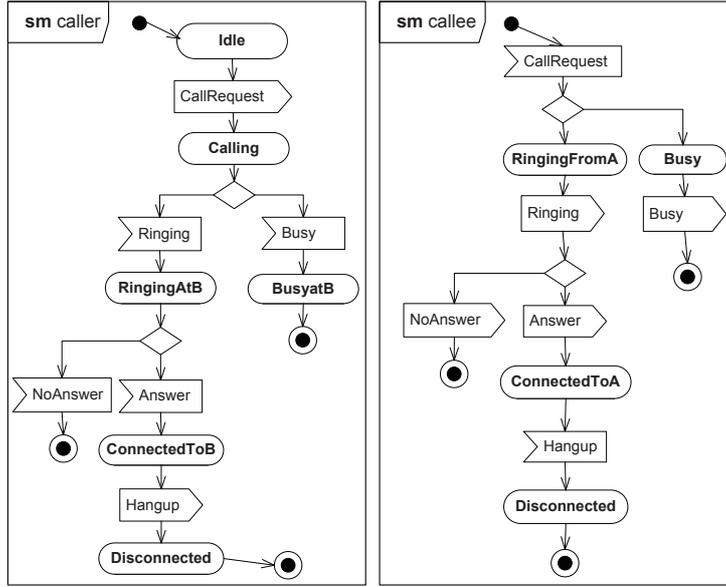


Figure 10: Role state machines for the `caller` and `callee` roles

The elementary role state machine for `callee` is generated from the `caller` lifeline of the `call` sequence diagram in Fig. 6, so that each message sent/received in the sequence diagram results in a send/receive signal in the state machine, and the three mandatory alternatives in the sequence diagram is reflected by three alternative paths in the state machine. Similarly, the elementary role state machine for `caller` is generated from the `callee` lifeline in the sequence diagram.

Step 5: Create the local PESM diagrams

Based on the transformation rules specified in [4]:

1. from the global PESM diagram, create one local PESM diagram for each composite role involved in the service (as given in step 1).
2. from the global policy rules, create the local policy rules to be associated with each PDP in the local PESM diagrams.

Again, this step should be applied recursively to each composite service specified as part of the composed service.

Step 5 applied to the tele-consultation service We illustrate this step by using the transformation rules specified in [4] to derive the local PESM diagram and local policy rules for the `PatientT` composite role shown in Fig. 11. The hierarchical structure of the `teleconsultation` global PESM diagram is preserved. The right side of Fig. 11 shows the `assisted caller` and `consulter` composite roles, derived from the aided `Registration` and `consultation` PESMs, respectively. As for the global PESM diagram, the policy rules to be associated with each PDP in the local PESM diagrams are illustrated as notes.

The local PESM diagrams for the `ReceptionistT`, `TeleSvcDsk` and `DoctorT` composite roles may be found in [8].

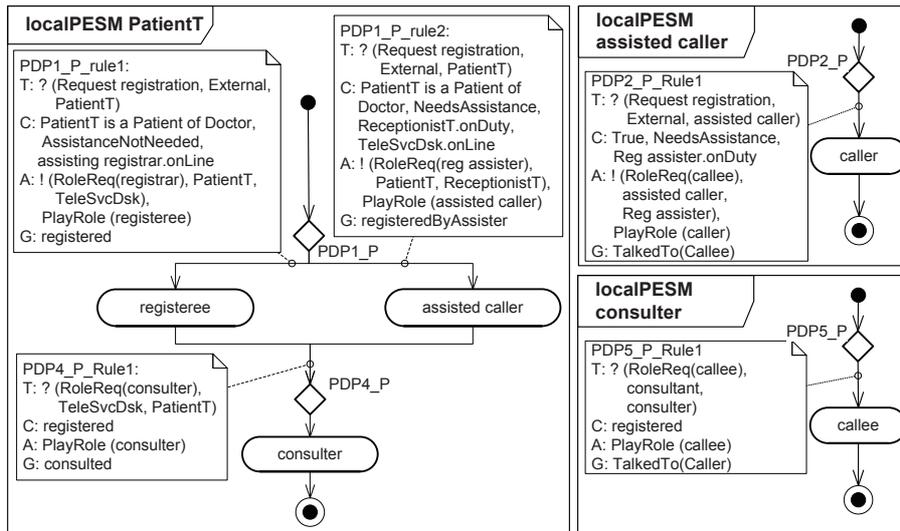


Figure 11: Local PESH diagram for the PatientT composite role

6 Methodology: The Realization Layer

At the realization layer, implementations are generated from the design models at the design model layer. In this section, we address the methodological steps performed in order to achieve this.

Step 6: Create and distribute service role implementations

For each elementary role state machine taken from the repository or created in step 4:

1. from the elementary role state machine, generate code if this is not already done. Such code generation may be performed automatically as described in [15,16].
2. distribute the relevant code to each of the subscribing components of the system. By subscribing components, we mean the system components interested in playing this role. Exactly which system components the roles should be distributed to is platform specific information and not part of the methodology described here.

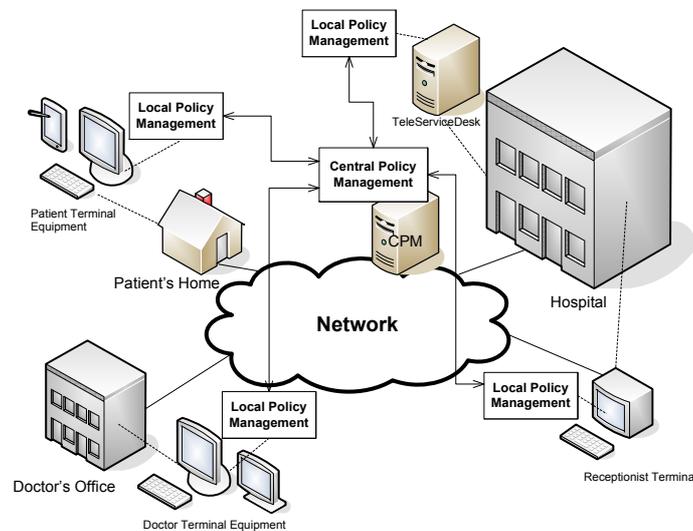


Figure 12: The policy managed tele-consultation deployment

Step 6 applied to the tele-consultation service A schematic diagram showing the tele-consultation system deployment is given in Fig. 12. The figure shows the distributed parts of the system: the patient terminal equipment located in the patient’s home, the doctor terminal equipment located at the doctor’s office, and the electronic tele-consultation service desk and the receptionist terminal equipment, each located in the hospital. Local policy management is deployed on each of the distributed parts of the system which are connected to the central policy management via the network. While not described in this paper, we have defined an architectural framework for a policy-based dynamic composition system, for information on this, we refer to [8].

For each elementary role state machine in Sect. 5, code is generated and distributed to each of the subscribing components. In this example, the code for the roles to be played by the `PatientT` composite role are distributed to the patient’s terminal, the roles to be played by the `ReceptionistT` composite role are distributed to the receptionist’s terminal, and the roles to be played by the `TeleSvcDsk` composite role are distributed to the tele-service desk terminal.

Step 7: Create and distribute PESM implementations

1. For each local PESM diagram from step 6:
 - (a) generate the local PESM description. This involves translating the control flow described by the PESM to a data format that can be used by the local policy management.
 - (b) distribute the description to the subscribing components.
2. For each of the local policy rules from step 6:
 - (a) translate the policy rules to a data format that can be used by the local policy management. A policy parser is required to do this. The parser takes the policy rule file and converts the rule to the data format required.
 - (b) distribute the rule to the subscribing components.
 - (c) extract the trigger event and distribute it to the subscribing components.

Step 8: Perform dynamic composition of the service

Dynamic composition is now enabled in the runtime system. This involves:

1. dynamic service execution, composing the distributed roles as governed by policy and PESM descriptions.
2. dynamic composition, e.g. involving the use of a user interface to change policy rules and include dynamic adaptation of services.

How steps 7 and 8 are actually done is platform specific and therefore we do not address this as part of this example. For a description of the policy-management system, and for results of investigations of feasibility of implementation of parts of the tele-consultation service and the some of the functionality of the policy management using the Arctis and Ramses tools suites [16] we refer to [8].

7 Conclusion

In this paper, a methodology is provided explaining how to use the policy-enabled concept to model, design and deploy dynamic composition of services. The

methodology enables a large degree of variability in the services that the provider is able to offer to customers. Policy is also used to enable dynamic choice between services to be offered. The methodology describes how to use the models to specify, design, and deploy services that may be adapted and managed more easily to suit the changing requirements of users and enable more rapid deployment of services.

References

1. Object Management Group: UML 2.2 Superstructure Specification, formal/2009-02-02. (2009)
2. Sanders, R.: Collaborations, semantic interfaces and service goals: a way forward for service engineering. Ph.D. thesis NTNU (2007)
3. Rossebø, J.E.Y., Bræk, R.: Using composition policies to manage authentication and authorization patterns and services. In: 3rd Intl. Conf. on Availability, Reliability and Security (ARES 2008), IEEE Comp. Soc. (2008) 597–603
4. Rossebø, J.E.Y., Runde, R.K.: Specifying service composition using UML 2.0 and composition policies. In: MoDELS 2008. Volume 5301 of LNCS., Springer (2008) 520 – 536
5. Castejón, H.N., Bræk, R.: Formalizing collaboration goal sequences for service choreography. In: FORTE'06. Volume 4229 of LNCS., Springer (2006) 275–291
6. Castejón, H.N., Bræk, R., von Bochmann, G.: Realizability of collaboration-based service specifications. In: Proc. of the Asia-Pacific Software Engineering Conference (APSEC 2007), IEEE Comp. Soc. (2007)
7. Funk, C., Schultheis, A., Linnhoff-Popien, C., Mitic, J., Kuhmunch, C.: Adaptation of composite services in pervasive computing environments. In: Proc. of the IEEE Intl. Conf. on Pervasive Services (ICPS'07), IEEE Comp. Soc. (2007) 242–249
8. Rossebø, J.E.Y., Bræk, R., Runde, R.K.: Methodology for policy-enabled dynamic composition of services. Technical Report AVANTEL 2/2009, Department of Telematics, NTNU (2009)
9. Floch, J., Bræk, R.: A compositional approach to service validation. In: SDL 2005. Volume 3530 of LNCS., Springer (2005) 281–297
10. Engelhardtson, F.B., Prinz, A.: Application of stuck-free conformance to service-role composition. Presented at the 5th Workshop on system analysis and modelling (SAM 2006) (June 2006)
11. Haugen, Ø., Stølen, K.: STAIRS — Steps to analyze interactions with refinement semantics. In: UML 2003. Volume 2863 of LNCS., Springer (2003) 388–402
12. Runde, R.K.: STAIRS — Understanding and Developing Specifications Expressed as UML Interaction Diagrams. PhD thesis, University of Oslo (2007)
13. Whittle, J., Schumann, J.: Generating statechart designs from scenarios. In: ICSE 2000: Proc. of the 22rd Intl. Conf. on Software Engineering, IEEE Comp. Soc. (2000) 314–323
14. Seehusen, F., Stølen, K.: Transformational approach to facilitate monitoring of high level policies. Technical Report All356, SINTEF Information and Communication Technologies (2008)
15. Kraemer, F.A.: Rapid service development for service frame. Master's thesis, University of Stuttgart (2003)
16. Kraemer, F.A.: Arctis and Ramses: Tool suites for rapid service engineering. Proc. NIK 2007 (Norsk informatikkonferanse) (2007)

Dependency-driven Parallel Programming*

Eva Burrows[†] Magne Haveraaen[‡]

Department of Informatics, University of Bergen, Norway

October 21, 2009

Abstract

With the appearance of low-cost, highly parallel hardware architectures, software portability between such architectures is in great demand. Software design lacks programming models to keep up with the continually increasing parallelism of today's hardware. This setting calls for alternative thinking in programming. When a computation has a static data-dependency pattern, extracting this pattern as a separate entity in a programming language, one can reformulate the computations. As a consequence, data-dependencies become active participants in the problem solving code. This allows us to deal with parallelism at a high-level. Data-dependency abstractions facilitate the mapping of computations to different hardware architecture without the need of rewriting the problem solving code. This in turn addresses portability and reusability issues.

1 Introduction

Computational devices are rapidly evolving into massively parallel systems. Multi-core processors are standard, and high performance processors such as the Cell processor [3] and graphics processing units (GPUs) featuring hundreds of on-chip processors, e.g. [8], are all developed to accumulate processing power. They make parallelism commonplace, not only the privilege of expensive high-end platforms. However, current parallel programming paradigms cannot readily exploit these highly parallel systems. In addition, each hardware architecture comes along with a new programming model and/or application programming interface (API). This makes the writing of portable, efficient parallel code difficult. As the number of processors per chip is expected to double every year over the next few years, entering parallel processing into the mass market, software needs to be parallelized and ported in an efficient way to massively parallel, possibly heterogeneous, architectures. Certain steps have been made towards standardization, e.g. OpenCL (Open Computing Language) has recently been announced as a framework for writing programs across heterogeneous platforms, and many hardware vendors seem to support it. However, the programming community is still in great need of high-level

*This paper recollects some of the theoretical results presented in an earlier work [2], by describing the underlying concepts at a higher level. In addition it also reports on the outcome of recent practical experiments.

[†]<http://www.iu.uib.no/~eva/>

[‡]<http://www.iu.uib.no/~magne/>

This paper was presented at the NIK-2009 conference; see <http://www.nik.no/>.