This report presents a general methodology for policy-enabled dynamic composition of services. The methodology focuses on how to model dynamic composition of services, and explains 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.

We provide an architectural framework for a policy-based dynamic composition system, and we explain how the service composition modelled at the service and design layers, can be deployed in the runtime system. In particular, the deployment of the policy management to enable dynamic service composition is explained.
Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Telematics
N-7491 Trondheim
Norway


ISBN 978-82-993980-7-7
ISSN 1503-4097
Abstract

This report presents a general methodology for policy-enabled dynamic composition of services. The methodology focuses on how to model dynamic composition of services, and explains 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. We provide an architectural framework for a policy-based dynamic composition system, and we explain how the service composition modelled at the service and design layers, can be deployed in the runtime system. In particular, the deployment of the policy management to enable dynamic service composition is explained.
Contents

1 Introduction 3

2 Background and Overview 5

3 Requirements 7
  3.1 The Architectural Framework 7
  3.2 The Methodology 7

4 The Architectural Framework 8
  4.1 Central Policy Management 9
  4.2 Local Policy Management 11

5 The Methodology 18

6 Case Study 21
  6.1 Description of the tele-consultation service 21
  6.2 Application of the methodology 22
  6.3 Specialising a Service 35
  6.4 Changing the Initial Specification by Adding Authentication 38

7 PESM and PDP execution 40
  7.1 Triggers vs. pins 40

8 Proof of Concept: Feasibility of Implementation 41
  8.1 Demonstration of Feasibility: Casting PESMs in Arctis 41
    8.1.1 Arctis adaptations 42
    8.1.2 Policy management, policy detection points and policies 42
    8.1.3 The Tele-consultation Service 44
  8.2 Discussion: Casting PESMs in Arctis 49

9 Discussion 49
  9.1 Implementation 50
  9.2 Tool support requirements 51
  9.3 Dynamic Service Composition 52

10 Conclusion 53

References 55
1 Introduction

A service is an identified functionality involving a collaboration among entities that achieves 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 as illustrated in Fig. 1.

The figure suggests an architecture for service-oriented systems, which is characterized by horizontal and vertical composition. On the horizontal axis, system components, are identified that are largely service independent and represent domain entities such as users, terminals, service enablers and shared facilities. They may reside in different computing environments. These domain entities such as users, user communities, terminals and resources are represented by actors in the system. On the vertical axis, several services and service components are identified (i.e. collaborations and collaboration roles) that depend on the system components of the architecture.

![Figure 1: Horizontal and vertical composition of services [2]](image)

We model a service in the context of UML 2.x [12], as a collaboration among roles with associated interactions behaviour. Service composition addresses the means of composing such collaborations, both statically and dynamically. Each collaboration is composed of elementary service collaborations, which are service collaborations with exactly two elementary roles that collaborate on an interface to provide a service or a service feature [19]. We specify the associated behaviour using UML 2.x sequence diagrams.

Service composition techniques can be categorized as static or dynamic, and manual or automatic. Static composition is performed at design or compile time, whereas dynamic composition is performed at runtime. However, there are varying degrees of dynamic composition that may be achieved depending on which dynamic composition techniques are used, as explained in [1, 13]. Manual composition means that the composition is performed by persons who have access to the elementary services whereas in automatic composition this is performed by a software agent, based on some predefined algorithms [6]. Automation of the service composition process means that either the ser-
vice composition technique employed can generate the choreography automatically, or if an abstract plan is provided, the service composition technique employed can locate the correct services to be composed [13]. We address the problem of dynamic composition of services in our work, leaving the automation problem for further work.

In [15], we introduced a policy-driven approach to service modelling, design, and implementation and we presented the concept of a composition policy and a policy enforcement state machine (PESM) for modelling the choreography of service collaborations. We have defined a composition policy as a set of policy rules specifying how a given set of collaborations can be composed to obtain a given set of composite services. A policy rule consists of four parts: trigger, condition, action and goal. In our approach, the PESM is defined statically (at design time) to govern the dynamic ordering of collaborations and policy-based choices among collaborations taken at runtime. These choices may involve dynamic coordination and negotiation among the components to coordinate dynamic role binding and role execution. The PESM diagram expresses graphically the ordering of policy decisions to be taken regarding ordering of collaborations when collaborations are composed dynamically. Each policy decision point (PDP) of the PESM identifies a set of policy rules. At the service layer, a global PESM diagram graphically depicts the dynamic composition of collaborations, at the design layer, local PESM diagrams graphically depict local coordination and enforcement of local composition policies and as such, depict a single component’s part in the dynamic composition.

In [17], we have specified the policy-based mechanism at the service and design model layers, providing a precise syntax for global composition policies at the service layer, and transformation rules for deriving local policies at the design model layer. In this report, we take a closer look at how the approach can be deployed in the runtime system, to enable dynamic service composition. This involves defining an architectural model for the policy-enabled dynamic composition system, and a process for runtime dynamic composition using the policy based approach.

For the architecture, we have been inspired by the Poema [11] approach, although in [11], the policy managed adaptation is used to do dynamic re-configuration of Mobile code applications specifically, and not to enable dynamic composition of services in general. The Chisel adaptive process described in [7] provides a guide for defining a process for runtime dynamic service composition. However, the actual way that the adaption is performed in Chisel involves manipulation of existing behaviour in the system, using metatypes to adapt existing objects. In our approach to dynamic service composition, we do not change the existing building blocks (elementary roles), but change which are combined together and in which order they are executed.

We present a methodology explaining how to use the policy-enabled concept to model, design and deploy dynamic composition of services. We exemplify the methodology using a tele-consultation service, which is based on the specification of the telemedicine consultation service described in [3]. We have simplified and changed the case description to keep the complexity of the system manageable, and to allow for addition of new behaviours to the service dynamically, to demonstrate the policy-based approach.

The main contributions of this report are:
an architecture for policy-enabled dynamic service composition
a methodology for applying the approach to dynamic composition of services; and
demonstration of the methodology for a tele-consultation case.

The rest of the report is organized as follows:
In Sect. 2 we provide background and an overview of the policy-enabled approach to
dynamic composition. Sets of requirements for the architectural framework and for the
methodology are given in Sect. 3. The architectural framework is presented in Sect. 4
and the general methodology for policy-enabled dynamic composition of services is
described in Sect. 5. In Sect. 6 we demonstrate practical application of the methodology to a
tele-consultation service. A proof of concept for realization of the policy-based dynamic
composition is provided in Sect. 8 followed by a discussion and evaluation Sect. 9 and
conclusion in Sect. 10.

2 Background and Overview

Figure 2 provides an overview of the policy-enabled approach to dynamic composition. At the service model layer, services are modelled structurally using UML 2.x collaboration diagrams [12], and policy rules stored in a table are used for governing dynamic composition. The policy enforcement state machine (PESM) 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 PESM 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 realization of the dynamic composition, which takes place in the runtime system. The policy-driven approach enables services to be dynamically executed, where ordering and choice between which services are composed is governed by policy. Services may be dynamically adapted to context, providing a service provider with flexibility when managing service deployments.

A policy-based dynamic composition approach requires a policy-specification model to express the composition strategies. We have chosen event-condition-action declarative policy rules [15, 17]. The main reasons for this choice are:

- in order to set up a service session, it is natural to use signalling. Signalling between components to set up a service session is an event-oriented activity. As such, an event-driven model is a natural candidate for notifying distributed components of changes that occur;
### Background and Overview

**Service Models**

<table>
<thead>
<tr>
<th>PESM</th>
<th>Policy Rules</th>
<th>UML Collaborations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="PESM Diagram" /></td>
<td><img src="image2" alt="Policy Rules Table" /></td>
<td><img src="image3" alt="UML Collaborations Diagram" /></td>
</tr>
</tbody>
</table>

**Policy Rules**

<table>
<thead>
<tr>
<th>Policy Rule ID</th>
<th>Condition</th>
<th>Trigger</th>
<th>Action</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>PESM1</td>
<td>P1 . A, B, C</td>
<td>R1</td>
<td>Do Out1</td>
<td>A, B, C</td>
</tr>
<tr>
<td>PESM2</td>
<td>P2 . A, C, D</td>
<td>R2</td>
<td>Do Out2</td>
<td>A, B, C, D</td>
</tr>
</tbody>
</table>

**UML Collaborations**

![UML Collaborations Diagram](image3)

---

**Design Models**

**Local PESMs**

<table>
<thead>
<tr>
<th>Local policy</th>
<th>Systems</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image4" alt="Local PESMs Diagram" /></td>
<td><img src="image5" alt="Systems Diagram" /></td>
<td><img src="image6" alt="Components Diagram" /></td>
</tr>
</tbody>
</table>

**Realization – Dynamic Composition**

![Realization Diagram](image7)

- declarative policies allow the designer/developer to specify dynamic composition of services from collaborations separately from the behavioural specifications of collaborations allowing for more freedom of choice in dynamic composition between which separately specified behaviours to compose.

In this approach, events specified as policy triggers are monitored, and these events are used to trigger the evaluation of one or more policy rules. We allow for policy rules with the same trigger to have overlapping conditions, so that more than one of them may be true at a given point in time. If this is the case, then the choice between which of the possible actions to be performed is made non-deterministically. This means, that if the conditions of a policy are matched, then the action specified by the policy rule may be executed.

A family of services will run dynamically in the system, according to a global PESM description and policy rules. But additionally, the policy rules and PESM diagrams may be changed by a central administrator. Changes made centrally are then distributed to the different parts of the system, and in this way services may be dynamically adapted and changed. Additionally, in the system, a system security policy, such as an availability pol-
icy may be violated causing a need to dynamically adapt the service. In such a case, the system would look up in a repository, the appropriated PESM and policies to apply. For example, if the management system registers $x \geq \text{maxlimit}$, the number of unauthorised uses of the service exceeds the limit of such unauthorised behaviour allowed by the availability management policy, this may trigger the requirement for stronger authentication to gain access to the service. In this case, the service must be dynamically adapted and composed with a sufficiently strong authentication pattern. Depending on which policy rules are satisfied different authentication patterns will be called.

3 Requirements

3.1 The Architectural Framework

As services will be distributed over several co-operating parts of a distributed system, in order to support policy-based dynamic composition of the distributed services, a management system is required. In this section we formulate a set of objectives for a policy-based dynamic composition management architecture.

- **The architecture should provide a means of supporting the realization of policy-based dynamic execution of services.** The approach to dynamic service execution outlined in Sect. 2 and described in [17], should be supported so that a developer/designer can model services and specify PESM diagrams and composition policies, and so that the design models derived according to the transformation rules specified in [17] can be transformed to code for deployment in the runtime system.

- **The architecture should support the separation of the functionality.** By this we mean that the policy-enforcement mechanism should be separated from the policies themselves, the monitoring function, and the services that are to be dynamically executed. This is important to provide a general distributed mechanism and to ensure consistency when triggering service compositions involving collaborations across collaborating computational objects deployed over different parts of the system.

3.2 The Methodology

In this section we formulate a set of objectives for the methodology.

- **The methodology should strive for easy adaptation in industrial development processes by being based on existing state-of-the-art UML methodology.** The methodology should provide a means of supporting the dynamic composition of patterns and services modelled using UML 2.x so that it may be widely used.

- **The methodology should explain how to model dynamic composition of services.** This involves describing how to use modelling techniques to specify, design, and deploy services that may be adapted and managed to suit the changing requirements of users and enable more rapid deployment of secure services (dynamically).
The methodology should be easy for a designer to understand and use. The methodology should be understandable to the developer/designer and increase the designer’s awareness of service availability issues while enabling the developer/designer to address the issues systematically through choice and specialization of e.g. an authentication pattern.

4 The Architectural Framework

The policy-management system is a separate management layer in the deployed service execution architecture. In [17], we explained how dynamic service composition can be modeled using UML 2.x and composition policies. A composition policy is a set of policy rules specifying how a given set of elementary collaborations can be composed to obtain a given set of composite services. As such, a composition policy applies to a family of related services, where the number of possible services in the family depends on the choices governed by the policy rules. Choices are made to decide which set of collaborations uses will be composed in order to deliver the selected service. Graphically, a global policy enforcement state machine (PESM) is used to depict the dynamic composition of collaborations according to the composition policy.

Policy-ruled dynamic composition of services is modeled by a designer using global PESM diagrams and for each PDP in the PESM, policy rules are specified, with one PESM for each family of services. From the global PESM and composition policy, local PESMs and local compositions policies can be generated, one for each composite role involved in the service. This is done according to the transformation rules given in [17]. Families of services that have been composed, compiled and deployed are executed according to the PESM description that applies to the family of services and the sets of policy rules that are referenced from the PDPs. When a developer/designer changes global composition policies or the global PESM diagram, this will change the executions of the family of services.

Additionally, the system may be monitored to ensure that service availability policies are enforced. An availability policy consists of an accessibility policy (e.g., required resources) and an exclusivity policy (e.g., which entities have permissions to use the service or system) [16]. When an availability policy is violated, an event may trigger an adaptation to the service family. At this point, there is a decision to adapt, and a different predefined PESM description is selected, based on a pre-defined availability policy, to satisfy the service availability requirements for the family of services. Availability policies can be edited by a system admininster.
4.1 Central Policy Management

The policy-based system architecture is modelled as a distributed architecture with a central management component and several local management components that are distributed to the different parts of the system. Fig. 3 shows the abstraction of the distributed architecture, showing the main parts: the Central Policy Management, and the Local Policy Management parts, one for each distributed part of the system. The Central Policy Management is responsible for distributing local policy rules and local PESMs to the distributed parts of the system. This is done routinely, as part of the operations support system. The Local Policy Management parts manage dynamic composition locally, negotiating with other local parts. The roles of the different parts of the architecture will be explained in more detail in following sections.

4.1 Central Policy Management

The central policy management consists of four components:

- *The Policy and PESM Specification Component*: The Global PESM diagrams and global composition policies may be entered into the system here using a tool-supported graphical interface. This is the component that directly interacts with the designer/developer. Whenever the designer/developer creates or modifies the PESM diagrams and composition policies, this is done using a tool supported graphical interface. A *Service Specification Tool* is used in combination with the policy and PESM specification component to specify services and their compositions. Details on how services and their compositions are specified may be found below in Sect. 5. The tool is supported by a *Collaborations and Roles Repository* where service specifications are stored for re-use.

- *Policy and PESM Control Component*: This centralized management service is responsible for activating global policies and global PESM descriptions for use in the distributed system, after they have been defined using the Policy and PESM Specification Service. For performing dynamic execution, both the policy rules and the PESM description are required. This component is responsible for distributing to the different parts of the system the local PESMs and local policies that have been generated from the global PESM and policies by the tool-supported environment. This service coordinates policy activation and deactivation, and is responsible for activation and deactivation of local policy rules and local PESMs as well as distributing this information to the different parts of the system.
4.1 Central Policy Management

- **Availability Policy Enforcement Component**: In this report, this is underspecified, and will be detailed in future work. This component is responsible for ensuring that the service availability requirements can be met. Exclusivity and accessibility properties may be specified by the designer for specific dynamically composed services. For example, it may be specified for a multimedia over IP (MMoIP) service, that the number of unauthorised accesses to the service should be limited, and, if the number of unauthorised accesses exceeds the pre-defined limit, then there is a decision to dynamically upgrade the authentication component of the MMoIP service.

- **Service Roles Control Component**: This centralized service is responsible for distributing to the local management the code that has been generated for each of the elementary service roles. A detailed explanation of how the elementary service roles are specified is given below in Sect. 5. Upon receiving the code for new elementary roles, the local management enters this into the local Service Role Store. Similarly, the centralized management is responsible for requesting deletions of roles by the distributed parts.

The central policy management is illustrated in Fig. 4 along with the service specification tool and the Collaborations and roles repository. Re-usable specifications are stored in the collaborations and roles repository. The service specification tool is used in combination with the policy and PESM specification service to specify services and their compositions. A detailed explanation of how these are specified is given in Sect. 5 below.
4.2 Local Policy Management

The code generated for each of the elementary role state machines is distributed to the local policy-based dynamic composition management services in the distributed parts of the system by the service roles control (SRC) component. This functionality is responsible for interfacing with the local management requesting the adding or deleting of elementary service roles code from the Service Role Store when new roles are centrally specified (or deleted) by a designer, and then code is generated for distribution to the different parts of the system, for use in dynamic service execution.

4.2 Local Policy Management

The local policy management is illustrated in Fig. 5. The local policy-based dynamic composition management consists of a set of managers which are involved in the dynamic service execution processes, a set of repositories, and the support for adding, deleting and changing items in the repositories. The repositories are the Event Store, the Local Policy Rule Store, the Local PESM Description Store, and the Service Role Store. In addition to the internal interfaces shown in Fig. 5, the Support Services have interfaces towards the central policy management, and the Event Manager and the RPM have external interfaces on links towards the other local parts of the system. The Availability Policy Manager also has external interfaces, although we do not specify these in this paper. Fig. 5 provides an abstract overview of the different components that are involved in the local policy management.

The management support services are needed so that the local policy management system is able to administer removals, additions, and changes to what is stored in the repositories, when notified to do so by the central policy management system. By this we mean that the central PESM and policy control component distributes all of the local PESMs, and local policies to the local management components. The management support services are responsible for:

- updating, adding, deleting policies in the Policy Rule Store, when notified to do so by the central control service. The central control service provides the instructions regarding existing rules, and provides the sets of rules to be added. The local support services then updates, adds, deletes policies in the Policy Rule Store;

- distributing named trigger events to the Event Store. These are the named local triggers that apply and that the Event Manager is required to listen for, in order to trigger dynamic role execution. For each PESM, a table of triggers to listen for is distributed to the Event Manager;

- updating, adding, deleting local PESM descriptions in the Local PESM Description Store, when notified to do so by the central policy and PESM control service, see Fig. 4. The central control service makes the changes to existing PESMs, and provides the PESMs to be added, and deletes PESMs that are no longer valid;

- adding or deleting elementary service roles from the Service Role Store when new roles are centrally specified (or deleted) by a designer, and then code is generated
for distribution to the different parts of the system, for use in dynamic service execution. Upon receiving roles to be added, and/or requests from the central control services to delete roles, the local support services ensures that the Service Role Store is up to date, by adding or deleting roles accordingly.

<table>
<thead>
<tr>
<th>Local Policy-based Dynamic Composition Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support (Local Events, Policy Rules, PESM descriptions, and Roles)</td>
</tr>
<tr>
<td>Event Store</td>
</tr>
<tr>
<td>Availability Policy Manager</td>
</tr>
<tr>
<td>Port towards multiple interfaces of the Central Management</td>
</tr>
<tr>
<td>Ports towards other local parts of the system, to listen for triggers</td>
</tr>
<tr>
<td>Port towards peer RPMs</td>
</tr>
</tbody>
</table>

Figure 5: Local Policy Management Architecture

The support services have external interfaces (input and output) towards the Central Policy Management Policy and PESM Control Services. It is envisioned that the support services also have external interfaces towards the centralized Availability Policy Enforcement Component, although we do not specify this fully in this paper.

The Event Manager, the Role Policy Manager, and the Service Role Session Manager interact to govern execution of dynamic service execution. Fig. 5 illustrates the Local Policy Management Architecture. The Role Policy Manager (RPM) is the active unit, interpreting PESMs and composition policies. In the following, the responsibilities of each of the managers are described:

- **The Event Manager**: The job of the Event Manager (EM) is to notify the RPM when a trigger happens. The EM is a stateless monitor that observes events that are sent and received across the local interfaces. The EM gets the list of which events to monitor from the event store. This list consists of all of the triggers given in the local policy rules tables for a local PESM. The EM does not handle stateful dependencies as these are specified in the condition part of a policy rule and are evaluated by the
RPM when a policy rule is evaluated. Based on the knowledge contained in the tables in the Event store regarding which events are to be monitored, the Event Manager listens for the events that trigger the start of dynamic service execution and events that occur during dynamic service execution. The EM notifies the Role Policy Manager (RPM) when a trigger has occurred.

- **The Role Policy Manager**: When the RPM receives a trigger notification message, the RPM is responsible for checking local policies and the associated local PESM description and negotiating with other local RPMs for dynamic composition of roles. Upon notification of an event, the RPM gets the policy rules with this trigger event that apply to the current PDP, according to the local PESM description. The RPM always knows where it is in a PESM diagram. By this we mean that the RPM always knows which PDP is current. Each PDP of a PESM has a pointer to a set of policy rules that apply at that point in time. Negotiation towards the peer RPM that manages the other collaborating role is required prior to signalling to the Service Role Session Manager (SRSM) that the role required can be played.

A trigger of a policy rule is a receive or send event of a message sent to or from a composite role. The RPM for this composite role will be the component discovering the trigger (upon notification of trigger occurrence by the EM) and responsible for enforcing the policy rule. A trigger is considered active by the RPM as long as a new trigger has not yet occurred on a particular lifeline of the PESM. This is particularly important for hierarchical PESM diagrams, where the same trigger is specified in the overall PESM and in the more detailed PESM diagram.

- **Service Role Session Manager**: The Service Role Session Manager (SRSM) is responsible for managing service sessions. Following a successful policy check at the first PDP for a local PESM by the RPM, the SRSM is notified that a service session can start and the SRSM starts the session and initiates the role play. Similarly, for each subsequent PDP that is passed during the dynamic execution.

- **Availability Policy Manager**: In this report, this managers role is underspecified, and will be detailed in future work. The APM is responsible for ensuring that the service availability requirements can be met by the local environment. The APM therefore notifies the RPM of availability policy violations, e.g. such as induced by context changes which effect the ability of the local environment to meet availability requirements. For example, if the component is hosted on a mobile terminal and the user roams to a network with poorer quality of service, the APM should notify the RPM of these changes.

The Event Manager (EM), the role policy manager (RPM), and the Service Role Session Manager (SRSM) are involved in the event monitoring, policy enforcement, and session management for a dynamic service execution. Fig. 6 depicts the steps involved in initiating a successful policy-based dynamic process. A trigger is either received from an external source, or a trigger is sent or received from an internal role. The Event Manager
(EM) knows which trigger events it needs to notify the RPM of, as it has been provided with a list of all triggers that the RPM should be notified of when they occur. The EM receives this list of events from the event store. The EM monitors (listens for) events and signals the RPM when such a trigger occurs. The RPM knows the PESM and the current PDP. Initially, when starting a new PESM, it is loaded from the local PESM Description Store. It then accesses policy rules with this trigger from the Local Policy Rule Store. The RPM checks a policy rule (PDPi_X_rulej). If the rule is satisfied, the peer RPM is notified and negotiation is carried out if needed for RolePlay. When notify/negotiation is successful the RPM signals the Service Role Session Manager (SRSM) to invoke the
role in the underlying service layer. The SRSM looks up the required role and the role is invoked.

As shown in Fig. 7, after the RPM has fetched the PESM from the PESM Store, it does not need to do so again each time it receives a trigger and examines a PDP. The RPM is responsible for keeping track of where the dynamic service composition is in relation to the PESM description, i.e., which PDP is the next PDP to evaluate. Upon receiving a trigger, the RPM checks the current PDP and gets the rules with this trigger that the PDP points to. The RPM checks that a policy rule with this trigger is satisfied. If a rule is satisfied, the peer RPM is notified and negotiation is carried out if needed for RolePlay. When notify/negotiation is successful the RPM signals the Service Role Session Manager (SRSM) to invoke the role in the underlying service layer. The SRSM
looks up the required role and the role is invoked.

The behaviour of the Role Policy Manager (RPM) involved in the dynamic process shown in Fig. 6 and Fig. 7, is elaborated on using a collaborative UML 2.x activity diagram. The activity diagram shown in Fig. 8, and detailed in Fig. 9 and Fig. 32 provides an overview of the distributed RPM behaviour. Fig. 8, shows the collaborative behaviour of two local RPM peers at a PDP of a PESM. This depicts the behaviour of the two local RPMs at the corresponding local PDPs and shows clearly the interactions between the local peers to coordinate start of the complementary roles. The diagram shows coordination of outputs so that two complementary roles may start.

The diagram shows the RPM for the initiating role, which is the role that either receives or sends the trigger, negotiate with the RPM for the participating role, which is the role that is triggered to participate in a role play. The behaviour of the RPM for the initiating role is shown to the left of the dividing line, and the behaviour of the RPM for the participating role is shown to the right of the dividing line in the figure. The RPM for the initiating role evaluates the policy rules with this trigger that the current PDP
refers to, and checks whether the rules are satisfied. For the rules that are satisfied, the operation Evaluate policy rules provides a list of roles that can start. This list is transferred with the RoleRequest to the participating role RPM. In the sub-activity Negotiate with peer, the entire list is evaluated. The participating role RPM either replies by confirming one of the roles or proposes a list of alternative roles. The initiating role RPM then checks the policies for the alternative roles, chooses one of the roles and confirms this particular role to the participating role, or sends a reject role message.

The sub-activity, Negotiate with peer, is shown in Fig. 9. As for Fig. 8, the behaviour of the RPM for the initiating role is shown to the left of the dividing line, and the behaviour of the RPM for the participating role is shown to the right of the dividing line in the figure. The RPM for the initiating role sends the role request message to the RPM for the participating role, containing the list of roles that can start. The RPM for the participating role gets the rules triggered by the role request message and in sub-activity PolicyDetectionPointEvaluation shown in Fig. 32, the participating role RPM checks which rules are satisfied, similarly to what was described above for the initiating role’s RPM. However, the participating role is allowed to propose alternative roles, based on the set of policy rules that the local PDP refers to, if the actor can not play the requested role. The negotiation exchanges for alternative sets of roles are shown.
5 The Methodology

In this section, we present the general methodology for policy-enabled dynamic composition of services. In the next section, we will demonstrate the practical usage of the methodology by applying it to an example of a tele-consultation service.

The methodology follows the three layered approach introduced in [15] and shown in Fig. 2. The methodology consists of the following 8 main steps:

1. **Service Model Layer: Specify the service (or family of services) using a UML 2.x collaboration overview diagram.**

   Using a UML 2.x collaboration overview diagram describe:

   (a) the distributed parts involved in the composed service, given as one composite role for each part.

   (b) the set of collaboration uses involved in the composed service. Each collaboration use is specified in one of the following ways:

   i. either, as an elementary collaboration, i.e. a collaboration between exactly two elementary roles,

   ii. or, as a composition of such elementary collaborations, where the composition is itself a sub-service (recursively) specified using a UML 2.x collaboration overview diagram in the same manner as described here.
(c) the binding of the (elementary or composite) roles in the collaboration uses to the composite roles from 1a.

This 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 the repository of re-usable service specifications depicted in Fig. 4. Each of the service specifications stored for re-use in the repository have been specified as collaborations according to 1b and for which the visible interface behaviour for each of the collaboration uses has been specified as in 2 (either 2a or 2b) below.

2. Service Model Layer: Specify the behaviour of each elementary collaboration used in the service.

For each of the elementary collaborations in 1b (both 1(b)i and 1(b)ii, if the collaboration and the behaviour of the collaboration is not already specified and stored in the repository of re-usable service specifications, specify the visible interface behaviour in one of the following ways:

(a) either, as a UML 2.x sequence diagram,

(b) or, using semantic interfaces [19].

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 [4, 5, 19].


Describe the choreography of the behaviour associated with the UML 2.x collaboration overview diagram from step 1 using

(a) a global PESM diagram for describing the ordering of the collaboration uses (see [17] for details on specifying PESM diagrams).

(b) a set of policy rules for each policy decision point (PDP) in the global PESM diagram. Together, these policy rules constitute the composition policy for the composed service.

If the diagram from step 1 uses sub-services (i.e. collaborations which are compositions of elementary collaboration) for which the behaviour is not already specified, their behaviour should be recursively specified using global PESM diagrams and composition policies as described here.
4. **Design Model Layer: Create one elementary role state machine for each elementary role used in the service.**

   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 [20, 21]. A semantic interface specifies the visible interface behaviour of role state machine, and is specified by an extended protocol state machine [19].

5. **Design Model Layer: Create the local PESM diagrams and local composition policies.**

   Based on the transformation rules specified in [17]:

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

6. **Realization Layer: Create and distribute implementations of the elementary role state machines.** For each elementary role state machine taken from the repository or created in step 4:

   (a) from the elementary role state machine, generate code if this is not already done. Such code generation may be performed automatically as described in [8, 9].
   
   (b) distribute the relevant code to each of the subscribing Local Service Role Store 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.

7. **Realization Layer: Create and distribute descriptions of the local PESM diagrams and local composition policies.**

   (a) For each local PESM diagram from step 5a:
   
   i. 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 Role Policy Manager.
   
   ii. distribute the description to the subscribing Local PESM Description Stores.
   
   (b) For each of the local policy rules from step 5b:
   
   i. translate the policy rules to a data format that can be used by the Role Policy Manager and the Event Manager. A policy parser is required to do this. The parser takes the policy rule file and converts the rule to the data format required.
ii. distribute the rule to the subscribing Local Policy Rule Stores.
iii. extract the trigger event and distribute it to the subscribing Local Event Stores.

8. **Realization Layer: Perform dynamic composition of the service.**

Dynamic composition is now enabled in the runtime system, as illustrated locally in Fig. 6. This involves:

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

The exact nature of the dynamic composition will be platform specific and is not described here. However, some concrete illustrations of how this step may be performed is given for the tele-consultation example in the next section.

The steps may be re-iterated at any time by the designer/developer to change or specialise the service.

Specialisation involves replacing a policy rule and the collaboration use it points to (that is, the collaboration use named in the action part of the policy rule specified during step 1 above) by a set of one or more policy rules and the collaboration uses named in the action parts of the policy rules. A policy rule and the collaboration use named by the policy rule is a specialisation if (1) the conditions of the condition part of the specialisation imply the conditions of the condition part of the policy rule being replaced and (2) if the role goals of goal part of the specialisation imply the role goals of the the goal part of the policy rule being replaced.

6  Case Study

In this section, we demonstrate the practical usage of the general methodology for policy-enabled dynamic composition of services given in Sect. 5 by applying it to an example of a tele-consultation service.

6.1  Description of the tele-consultation service

The tele-consultation service that we use as an example to demonstrate the practical usage of the methodology is based on the specification of the telemedicine consultation service described in [3]. We have simplified and changed the case description to keep the complexity of the system manageable, and to allow for addition of new behaviours to the services dynamically, to demonstrate the policy-based approach to dynamic service composition. The roles involved in a tele-consultation are a patient, a receptionist, and a doctor, and the electronic tele-consultation service desk.
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 (or hospital) for routine consultations when routine tests may 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 the 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, the human receptionist may be contacted. The human receptionist then gets the registration information from the human patient, and provides this information to the electronic tele-consultation service desk.

![Collaboration overview diagram for the tele-consultation case](image)

**Figure 11: Collaboration overview diagram for the tele-consultation case**

### 6.2 Application of the methodology

We apply the 8 main steps of the methodology as follows:

1. **Service Model Layer**: Specify the tele-consultation service (or family of services) using a UML 2.x collaboration overview diagram.
In this step use a UML 2.x collaboration overview diagram to describe the tele-consultation service statically and structurally as shown in Fig. 11. In order to arrive at this collaboration overview we proceed as follows:

(a) the distributed parts involved in the composed service are represented by the composite roles PatientT, ReceptionistT, DoctorT and TeleSvc-Dsk.

(b) the set of collaboration uses involved in the composed service are specified as follows:

i. as elementary collaborations, as shown in Fig. 12. The registration collaboration models registration of a patient’s request for consultation with a doctor. The call collaboration models a two way communication between two parties such as a telephone call. The call collaboration is used e.g. when a patient requires assistance to register.

ii. as compositions of elementary collaborations, as shown in Fig. 13 and Fig. 14. The aidedRegistration collaboration is invoked when a human patient requires special assistance to register. In this case, the human receptionist is called so that the human patient can provide the registration to the human receptionist. The human receptionist obtains the registration information from the human patient during the call, and provides this information to the electronic tele-consultation service desk. The consultation collaboration is invoked when a human patient has registered and the human doctor is available for a teleconsultation. The doctor is informed by the tele-consultation service desk that the patient
6.2 Application of the methodology

Figure 13: The aided Registration collaboration

Figure 14: The consultation collaboration

is waiting for a consultation. The doctor calls the patient and is able to access the patient journal via the tele-consultation service desk.

(c) the binding of the (elementary or composite) roles in the collaboration uses to the composite roles from 1a is shown in Fig. 11. The teleconsultation service is composed of three collaboration uses. The collaboration use \texttt{prs} of the registration collaboration is an elementary collaboration for which the elementary roles \texttt{registeree} and \texttt{registrar} are bound to the \texttt{PatientT}
and TeleSvcDsk composite roles, respectively. The collaboration use pcrs of the aidedRegistration collaboration is a composite collaboration, for which the registration collaboration is reused, in collaboration use rrs, for which the elementary roles registeree and registrar are bound to the reg assister and assisting registrar roles, respectively.

2. Service Model Layer: Specify the behaviour of each elementary collaboration used in the service.

We illustrate this step by specifying the UML 2.x sequence diagrams for the registration and call collaboration as described in 2a. The UML 2.x sequence diagram for the registration collaboration is shown in Fig. 15 and the UML 2.x sequence diagram for the call collaboration is shown in Fig. 16. In both of these sequence diagrams, we use the STAIRS [18] operator xalt to express that each of the alternatives are mandatory in the final implementation.

---

**Figure 15: UML 2.x sequence diagram for the registration collaboration**

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, please see [19] which uses a very similar call specification to demonstrate the validation techniques.
3. **Service Model Layer:** Specify the composed service behaviour using a global PESM diagram together with a global composition policy.

The choreography of the behaviour associated with the UML 2.x collaboration overview diagram shown in Fig. 11 from step 1 is described in the global PESM diagram given in Fig. 17. For the convenience of the reader, the policy rules are shown as notes annotated to the branches of the PESM diagram, one for each policy rule. As explained above in step 1, the tele-consultation service uses two sub-services, the aided registration service and the consultation service.

As the collaboration overview diagram shown in Fig. 11 from step 1 uses the sub-service aided Registration which is a composition of elementary collaborations, the behaviour is recursively specified using a global PESM diagram and composition policies as shown in Fig. 18. During aided Registration, the
6.2 Application of the methodology

As can be seen in Fig. 11 and Fig. 18, the trigger of PDP1_Rule2 in the main PESM diagram of Fig. 11 follows PDP1 and is the trigger for PDP1_Rule2 in the detailed diagram shown in Fig. 18. When the trigger has happened, and the conditions of PDP2_Rule1 are true, then the action is that the call collaboration is performed. Evaluation of rule 1 at PDP3 is triggered when the Answer message of the call collaboration is sent by the ReceptionistT to the PatientT.

In order for the goal of policy rule PDP1_Rule2 of Fig. 17 to be true, each of the role goals of the policy rules PDP2_Rule 1 and PDP3_Rule1 shown in Fig. 18 must also be true. This means, for example, that the role goal ReceptionistT.assistedRegistrationComplete of PDP1_Rule2 is true when
the role goals Receptionist\text{\textit{T}}.called and Receptionist\text{\textit{T}}.Patient-Registered of PDP2\_Rule1 and PDP3\_Rule1, respectively, both are true.

As the collaboration overview diagram shown in Fig. 11 from step 1 uses the sub-service consultation which is a composition of elementary collaborations, the behaviour is recursively specified using a global PESM diagram and composition policies as shown in Fig. 19. During consultation collaboration, the doctor interacts with the electronic tele-consultation service desk and the doctor is in a call with the patient. By interacting with the electronic tele-consultation service desk, the doctor is able to obtain patient data as well as record information in the patient journal.

The policy trigger for each of the policies that PDP4 points to in the main diagram for the global PESM diagram for the tele-consultation service given in Fig. 17 follows PDP6 in the subordinate PESM for the consultation service. As can be seen by the diagrams, the trigger for PDP4\_Rule1 in the main diagram is the same as the trigger for PDP6\_Rule1 in the detailed PESM diagram for the consultation composite service. Similarly, the trigger for PDP4\_Rule2 in the main diagram is the
same as the trigger for PDP6_Rule2 in the detailed PESM diagram. When one of the two rules PDP6_Rule1 or PDP6_Rule2 applies, the administration collaboration can be performed. As part of the behaviour of the administration collaboration, the TelesvcDsk then triggers the evaluation of policy Rule1 at PDP5 by sending the StartConsultation message to the DoctorT. 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 by the DoctorT and the condition part of the policy rule is true.

Figure 19: Global PESM diagram for the consultation collaboration
4. **Design Model Layer**: Create one elementary role state machine for each elementary role used in the service.

For each elementary collaboration not taken from the repository but specified in step 2, one elementary role state machine is created for each of the two roles involved in the collaboration. Fig. 20 shows the elementary role state machines for the **caller** and **callee** roles involved in the **call** collaboration.

![Role state machines for caller and callee roles](image)

**Figure 20**: Role state machines for the **caller** and **callee** roles
5. **Design Model Layer**: Create the local PESM diagrams and local composition policies.

Based on the transformation rules specified in [17], the local PESM diagrams and local policy rules are derived for each the composite roles involved in the tele-consultation service. In Fig. 21 the local PESM diagram for the **PatientT** composite role is shown. The local PESM diagrams for the **ReceptionistT**, **Tele-SvcDsk** and **DoctorT** composite roles are shown in Fig. 22, Fig. 23 and Fig. 24, 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.

![Local PESM diagram for the PatientT composite role](image)

*Figure 21: Local PESM diagram for the PatientT composite role*
Figure 22: Local PESM diagram for the ReceptionistT composite role
6.2 Application of the methodology

Figure 23: Local PESM diagram for the TeleSvcDsk composite role

Figure 24: Local PESM diagram for the DoctorT composite role
6. **Realization Layer:** Create and distribute implementations of the elementary role state machines.

A schematic diagram showing the tele-consultation system deployment is given in Fig. 25. 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.

![Schematic showing the policy managed tele-consultation deployment](image)

**Figure 25:** Schematic showing the policy managed tele-consultation deployment

For each elementary role state machine created in step 4:

(a) from the elementary role state machine, code is generated as described in [8, 9].

(b) the relevant code is distributed by the central management system to each of the subscribing Local Service Role Store components of the system. 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 ReceptionisT 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
6.3 Specialising a Service

desk terminal.

7. **Realization Layer: Create and distribute descriptions of the local PESM diagrams and local composition policies.**

   How this is actually done is platform specific/implementation dependant and therefore we do not address this as part of this example.

8. **Realization Layer: Perform dynamic composition of the service.**

   How this is actually done is platform specific/implementation dependant and therefore we do not address this as part of this example.

In the next sections we provide examples of how the tele-consultation service can be specialised and changed.

### 6.3 Specialising a Service

In this section, we exemplify specialisation of a service. For any of the sub collaborations specified in the UML 2.x collaboration static structure specification of a service of step 1, the sub collaboration may be specialised to allow for choice in dynamic composition. This is done during step 1, or during a reiteration of step 1. By introducing specialisations as a reiteration of step 1, the 8 steps are then reiterated also.

For example, for the tele-consultation case, in the sub-collaboration use `pcr` of the `call` collaboration, the designer may specify a choice between three specialisations: `chat`, `voice`, or `video`. Depending on conditions specified in the composition policies, such as type of terminal support at the Patient, one of these three specialisations will be chosen during dynamic composition of the service. In Fig. 26 the PESM diagram has been extended to include the specialisation of the collaboration use `pcr` of the `call` collaboration, to a choice between the collaboration use `pcr1` of the `chat` collaboration, collaboration use `pcr2` of the `voice` collaboration, and collaboration use `pcr3` of the `video` collaboration. This is done using the tool, by replacing the single collaboration with the three specialisations, and replacing the original policy with three new policy rules, one for each specialisation. We require that the specialised policy rules fulfill all of the obligations of the policy rule that is being replaced. The resulting, revised global PESM diagram for the `aidedRegistration` collaboration is shown in Fig. 26.

The three new rules, `PRP2_Rule1s`, `PRP2_Rule2s` and `PRP2_Rule3s` shown in Fig. 26 are all specialisations of the policy rule `PRP2_Rule1` for `PDP2` of the global PESM diagram shown in Fig. 18 of Sect. 6.2. The conditions of the condition part of the specialisations imply the conditions of the condition part of the policy rule being replaced as each of the conditions of policy rule `PRP2_Rule1` are also conditions of the three new rules, `PRP2_Rule1s`, `PRP2_Rule2s` and `PRP2_Rule3s`. Similarly, the role goals of goal part of the specialisations imply the role goals of the the goal part of the policy rule being replaced.
In Fig. 27, the specialisation of the local PESM diagram for the assisted caller composite role is shown, as generated from the global PESM and composition policies using the transformation rules.
6.3 Specialising a Service

Figure 27: Local PESM diagram for the assisted caller composite role used in the aided Registration collaboration with specialisations of the call collaboration.
6.4 Changing the Initial Specification by Adding Authentication

Consider, for example, for the Tele-consultation case, that a security analysis recommends that authentication is required for direct registration to the electronic tele-consultation service desk. The analysis has determined that authentication is not required, for aided registration, as the patient speaks to the receptionist, and not directly to the electronic tele-consultation service desk. We add therefore, the unilateral two pass authentication (UTPA) collaboration to the teleconsultation collaboration as shown in Fig. 28. This is then added to the global PESM diagram prior to the collaboration use prs of registration in the Teleconsultation PESM for the tele-consultation case, as shown in Fig. 29.

For direct registration to the electronic tele-consultation service desk, authentication is required. We add therefore authentication to the PESM, changing the Global PESM. For our simplified case, we consider direct authentication to the electronic tele-consultation service desk to be sufficient. However, if the case is elaborated so that patients are allowed to browse information at the hospital such as a patient journal or lab results, then different access rights will need to be allocated on a per patient basis. Support for this requires e.g. the User Pull authentication and authorisation patterns. There should be tool
support consisting of a repository that stores sets of authentication and authorisation patterns for re-use. For more information on choice of which AA-patterns for composition with services, please see [14].

Figure 29: Global PESM for the tele-consultation service, with authentication added
7 PESM and PDP execution

Before providing a discussion of the feasibility of implementation of the policy-based approach, in this section we take a closer look at global PESM diagrams in order to give a precise understanding of the behaviour associated with the diagrams.

A global PESM defines the combined behavior of policy management and service execution in terms of service collaborations and service roles. It has two main types of nodes that are linked by token flows in the fashion of UML activity diagrams:

- Policy decision points (PDP) that uses policy rules to govern branching in the PESM;
- Service activity nodes, which in global PESMs represents service collaborations and in local PESMs represents service roles.

Each PDP refers to a set of policy rules that are evaluated when a token enters the PDP node. The token is then passed on to an outgoing branch corresponding to which policy rules that evaluate to true. In this manner the PESM precisely defines an ordering of PDPs and service activity nodes. During execution of a PESM, the policy rules to evaluate are identified by the current PDP.

For the trigger part of policy rules one must consider two cases:

- The trigger is internal to a service activity node (which represents a collaboration in a global PESM and a role in a local PESM) and has caused a token to flow from the activity node into the succeeding PDP. One may assume that the token carries information about the trigger and thus enables the PDP to know which trigger has occurred in cases where several alternatives are possible.

- The trigger is external to the preceding service activity nodes, and thus must be captured and signaled to the PDP by means of separate event monitoring mechanisms. This may involve waiting in the PDP until the external event has occurred.

In both cases it is assumed that the trigger is known to the PDP when policy evaluation is performed.

7.1 Triggers vs. pins

As mentioned above, some triggers are events of the interactions taking place inside service collaborations represented as service activity nodes in the PESM. Other triggers specify events of interactions that are external to the service activity nodes of a PESM. They are typically occurring on external interfaces or in collaborations not modeled in the PESM. We will use the terms internal trigger and external trigger to distinguish the two cases.

When the token leaves a service activity node of a PESM, all events that may be used as internal triggers in the following PDP have already occurred and can immediately be
used to evaluate its policy rules. The question is how to capture the events and pass
the information on to the PDP. From a modeling point of view, the simplest is to use
the standard mechanisms of activity diagrams; either to pass the information along with
the token emitted from the service activity node, or to use different parameter output
pins to represent different triggers (i.e. different outcomes) of a service activity node.
This latter option is not used in the present PESMs, but is demonstrated in the proof-
of-concept experiment explained in Sect. 8. One attractive property of pins is that they
hide interaction details inside service activity nodes, and offer an interface to service and
policy developers that is independent of the interaction protocol.

In the Architectural framework, explained in Sect. 4, one assumes that the implementa-
tion is based on event monitors that listen to the interactions taking place among the
roles of a collaboration and capture triggering events. When a trigger occurs, it is fed to
the PDP, which will then evaluate the policy rules. It is assumed that the event monitoring
is stateless, and that state dependency is expressed in the condition part of policy rules.
With the architectural framework described in Sect. 4, there is no difference in the way
that internal and external triggers are handled by the role policy manager at a PDP, except
that one may have to wait in the PDP until external event occurs.

External triggers require that external events are monitored and captured by something
external to the service activity nodes of the PESM.

8 Proof of Concept: Feasibility of Implementation

In this section we consider the feasibility of implementation of the policy-driven approach
to dynamic service composition. The aim is to demonstrate the new principles, not to
make a new (complete, integrated) tool. We therefore use existing tools for a proof of con-
cept study of the policy-driven approach to dynamic service composition. In Sect. 8.1 we
present the investigations of feasibility of implementation of parts of the tele-consultation
service and the RPM functionality of the architectural framework using the Arctis and
Ramses tools suites [9]. In Sect. 9.1 we provide a summary description of what needs
to be done in order to provide a full implementation of the policy-based approach. In
Sect. 9.2, we provide a set of requirements for a complete, integrated tool.

8.1 Demonstration of Feasibility: Casting PESMs in Arctis

Arctis [9] is a tool for UML2 collaboration and activity diagrams intended to model ser-
VICES as collaborations and collaboration compositions. Services are modeled statically
in terms of roles and sub-collaborations using collaboration diagrams. The behavior of
collaborations is defined using activity diagrams where roles are represented by parti-
tions. In this way it is possible to define service behaviors completely with support for
compositional model checking and automatic synthesis of role state machines from which
executable code may be automatically generated.

We asked ourselves if it would be possible to demonstrate the feasibility of our ap-
proach and its potential for tool support by casting the policy management and PESM
diagrams in Arctis? In the following sections, we present what we found out.

### 8.1.1 Arctis adaptations

In order to fully model what goes on when a PESM is executed, one needs to model the combined behavior of the policy management and service execution. In the system architecture, described in Sect. 4, the service management is provided in a layer separated from service execution. Since there is no support for such layering in Arctis the two layers had to be "flattened" when modeled in Arctis.

All control flows must be local to a partition, i.e. to a role, in Arctis. One cannot specify non-localized flows as we do in the global PESMs. Therefore the Arctis models are suitable to provide a global view of the distributed behavior made up of local PESMs, PDPs and service roles. This means that a PDP is modeled as a collaboration performing distributed service management and not just as a decision as in the PESM diagrams.

Rather than using interaction events directly as triggers one either uses parameter output pins to represent different outcomes of an activity, or lets the information be carried by the token leaving the activity. It is necessary to make a distinction between internal triggers (events that occur as part of a collaboration), and external triggers (triggering events that are external to the modeled collaborations, e.g. a user interaction). Internal triggers are encapsulated by the collaboration activity and mapped to output pins, while external events are captured by external event monitoring activities.

### 8.1.2 Policy management, policy detection points and policies

A global view of collaboration between two distributed PDPs is modeled as shown in Fig. 30.

![Figure 30: Collaboration between two distributed PDPs](image-url)
In Fig. 31, it is assumed that internal triggers are represented by the token flows leading to PDPs. Thus the input pin represents a known outcome of the preceding collaboration, and implicitly an internal trigger if needed. If an external trigger is needed, an external monitoring activity is initiated by issuing a token through the streaming pin Get-ExternalEvent. When the external event occurs, the event monitor returns a token through the streaming pin Event to signal this. Once the triggering event has occurred, be it external or internal, the next step is to evaluate the PDP rules for this trigger locally in the initiating role. This will result in one or more rules evaluating to true, with corresponding actions in terms of sub-roles to be proposed to a participating role. There may be several alternative participating roles, but negotiation will only be carried out with one at the time. For instance in the tele-consultation service, the participating role in Registration and AidedRegistration are different.

Since Arctis assumes that data structures and operations are defined using Java, one must use Java to define the policy rules and policy decisions. This has not yet been elaborated.

In Arctis, elementary collaborations are modeled as activities where interactions are represented by flows that cross partition boundaries. Such an activity can be used as a (distributed) building block that encapsulate the interactions and only offer pins to its
environment, as illustrated by the Negotiate with Peer activity in Fig. 32.

The NegotiateWithPeer collaboration starts with a role request that carries a (set of) proposed Role(s). The participating role then evaluates its local PDP rules in activity PolicyDetectionPointEvaluation and either agrees to (one of) the proposed role(s) or proposes an alternative role to the initiating role which is then considered in the activity CheckProposalPolicies. The outcome is either a confirmation that both agree on the role, or a rejection which means that they do not agree.

8.1.3 The Tele-consultation Service

The tele-consultation service is modeled in Arctis as shown in Fig. 33. It depicts the same as in Fig. 17, with some differences:

- The PDPs are represented as collaborations and therefore they are visually different from those in Fig. 17.
- The composite roles are represented as partitions.
- Parameter output pins are used to represent internal triggers.
- Arctis supports streaming pins which can be used to model interactions between concurrent activities more precisely than in PESMs, here between PDPs and external event monitoring.
8.1 Demonstration of Feasibility: Casting PESMs in Arctis

- All control flows are local to partitions (i.e. composite roles). Semantically this is the biggest difference, but in accordance with the mapping to local PESMs.

![Figure 33: Tele-consulation modelled in Arctis](image)

The PDP2 activity of Fig. 17 is detailed in Fig. 34. PDP2 contains two instances of NegotiateWithPeer. It was necessary in Arctis to model two instances of NegotiateWithPeer to model the policy determined choice between the Registration and AidedRegistration since these involve different participating roles.

The AidedRegistration is detailed in Fig. 35 and the Consultation in Fig. 36.
8.1 Demonstration of Feasibility: Casting PESMs in Arctis

Figure 34: Activity diagram showing the PDP2 collaborative behaviour

Figure 35: Activity for the Aided Registration collaborative behaviour
The Call collaboration is used in both the AidedRegistration and Consultation collaborations. The behaviour associated with the Call collaboration is shown in Fig. 37. The Call Set Up is detailed in Fig. 38 and the Call Terminate is detailed in Fig. 39. Note that the flow lines crossing partition boundaries represent interactions. When Arctis synthesizes state machines for the roles, it will automatically create messages to be passed over a buffered communication medium. The message names thus created will not necessarily correspond to the messages of the Sequence diagrams in Fig. 15 and Fig. 16.
8.1 Demonstration of Feasibility: Casting PESMs in Arctis

Figure 38: Activity for the Call Setup collaborative behaviour

Figure 39: Activity for the Call Terminate collaborative behaviour
8.2 Discussion: Casting PESMs in Arctis

When casting the PESM diagrams in the Arctis tool [8, 9] we were forced to carefully consider all details of the PESM diagrams from a slightly different point of view and to make sure the resulting models were passing the static checks performed by the tool. Although our explorations using the Arctis tool are not complete in all details, they served to demonstrate feasibility and basic soundness. The main lessons learned from this experiment are the following:

1. It is possible, with some adaptation, to represent the behavior of PESMs and their distributed PDPs precisely using UML 2.x collaborations and Arctis.

2. Thus basic tool support for the localized flows of PESMs is feasible by means of Arctis.

3. In the present Arctis models, the policies need to be defined using Java.

4. More elaborate support for editing and validating policies will require additional functionality. It seems likely that such functionality can be provided as plug-ins to Arctis.

5. The Arctis models policy management and service collaborations in essentially the same way without separating these in different layers. To fully realize the architectural solution from Sect. 4 one would need to develop a platform support that clearly separates service roles from policy management roles.

6. Support for realising clear separation of service roles from policy management roles in the tool deserves further attention in future work.

Arctis offers two important features that have not been utilized in the present experiment, model checking to ensure dynamic correctness of the models, and design synthesis to generate executable implementations from the models. This will be explored in future work.

9 Discussion

In this report, we have outlined how the policy-enabled dynamic composition of services can be implemented. We have exemplified use of the method for the tele-consultation case, however, the method is first practical and usable with tool support. Although we have not fully implemented the tele-consultation service presented, and there is currently only basic tool support for specifying PESM diagrams, we claim that it is quite feasible to implement the approach, based on our explorations using the Arctis tool [8, 9], as explained above in Sect. 8.2.
9.1 Implementation

An implementation project to implement the Policy-based Dynamic Composition approach and methodology will consist of two parts, 1) implementation of tool support for specifying PESMs and composition policies, and 2) Implementation of policy management support in the runtime system, to enable execution of services dynamically.

1. Implementation of tool support

- **Service model layer**: Implementation of tool support for specifying global PESMs and policies. The tool should also support specification of Collaborations hierarchically, where the basis modelling element is an elementary two-party collaboration. For each elementary collaboration, the visible interface behaviour is defined e.g. by a UML sequence diagram or semantic interfaces. The tool should provide support for validation of the visible interface behaviour between the two roles of an elementary collaboration with respect to safety and liveness properties as described in [4, 5, 19].

- **Design model layer**: Implementation of transformation rules for transforming the global PESMs and policies to local PESMs and local policies (this should be straightforward, as these rules are specified in [17]).

- **Realization layer**: Code generation. This includes generation of code for each of the elementary role state machines, as well as machine executable local policy rules and local PESM descriptions.

2. Implementing the management system. Chisel [7] and POEMA [11] are similar examples of such a policy-based management system, although they are used in a different way (as explained in the Sect.1 above). Both Chisel [7] and POEMA [11] are examples that demonstrate that implementing such a policy-based approach to managing dynamic execution of services is possible.

Working the tele-consultation example to demonstrate the practicality of the methodology without tool support (other than a visio template for UML 2.0) was very tedious as it was difficult to ensure consistency across the different diagrams. A tool should include support for ensuring consistency across the different service layer specifications (the UML 2.x collaboration overview diagram, the sub-services and elementary collaborations, the interface behaviour specifications for the elementary roles, the PESM diagram and the global composition policies).

For example, to ensure consistency between the PESM diagram and the composition policy, the tool should support automatic generation of PDP name for each PDP within a global PESM diagram. Additionally, each policy rule name should be automatically generated, including the pointer to the PDP. When the PDP is clicked on to create rules, the name should be automatically generated, and, the rule template should pop-up with syntax for each of the rule fields (T, C, A, G) to guide the designer to correctly complete the rule. Regarding the goal part of the policy, this should specify (or refer to) the role
9.2 Tool support requirements

Based on the Case Study of Sect. 6 and the Proof of Concept investigations for realization of the policy-based dynamic composition approach of Sect. 8.1, we arrive at a set of requirements for tool support. Tool support is required for:

- **specifying the service using a UML 2.x collaboration diagram overview as described in step 1 of the methodology.** This includes support for specifying each of the collaboration uses involved in the composed service along with support for specification of the behaviour of each elementary collaboration used in the service as described in step 2 of the methodology. Additionally, the tool should provide support for validation of the visible interface behaviour between the two roles of an elementary collaboration with respect to safety and liveness properties as described in [4, 5, 19].

- **a repository/library for storing re-usable service specifications specified as described in steps 1 and 2 of the methodology.** It should be possible for the developer to access the repository/library and select such service specifications for re-use when specifying a service (or family of services).

- **specifying the global PESM diagram and composition policies for the service (or family of services) as described in 3.** Support is required for defining, browsing and editing policies that point to the PDPs of the PESM. To ensure consistency, the PESM specification should be linked to the UML 2.x collaboration overview diagram. This may be done by clicking on a collaboration use in the UML 2.x collaboration overview diagram to select ordering/placement of a collaboration use as a node in the PESM.

- **creating the design model specifications as described in steps 4 and 5 of the methodology.** It is desirable that the tool should support automatic transformations to local policies and local PESMs.

- **creating and distributing implementations of the elementary role state machines, the local PESM descriptions and the local composition policies as specified in steps 6 and 7 of the methodology.** The resulting policies should be machine-executable,
for use by the RPM in governing dynamic service execution. Similarly, the local PESM descriptions should be machine-executable for use by the RPM.

9.3 Dynamic Service Composition

In Sect. 1 we pointed out that there are varying degrees of dynamic composition that may be achieved depending on which dynamic composition techniques are used, as explained in [1, 13]. In this section we explain the degree of dynamic composition that can be achieved using the methodology presented in this report. We provide this discussion by dividing the degree of dynamic composition that can be achieved into four levels as stated and explained in the following:

1. **Dynamic choice in service execution.** At this level, the complete PESM and the composition policies its PDPs point to is specified at design time, policy only is used during service execution to select paths/subgraphs. This level of dynamic composition can be achieved by following each of the steps of the methodology provided in Sect. 5. At runtime, services are executed dynamically, depending on policy choices. Policy governs which services are combined at runtime and in which order, but according to the pre-defined plan specified by the PESM and composition policies.

2. **Dynamic choice in service execution with policy induced changes.** In addition to 1., the addition and/or removal of pre-defined subgraphs is enabled by changing policy rules (adding/changing/removing policy rules). This level of dynamic composition can be achieved by reiterating the steps of the methodology provided in Sect. 5, resulting in changes to the PESM and policy changes that need to be distributed to the system as well as possible additions/changes/deletions to service roles available in the distributed parts of the system.

3. **Dynamic choice in service execution with policy induced changes and service plug-ins.** In addition to 1. and 2., it is possible to plug-in collaborations dynamically into the PESM diagram by matching the goals of collaborations to the goals stated in the policy rules. This level would allow for parts of a PESM to be further specialised for different situations. For example, not just specifying unilateral two pass (UTPA) authentication as the authentication collaboration to use, but also allowing a range of different specializations of UTPA to be plugged in, depending on the situation. While we have not addressed this level directly in our work, it is fully possible by extending the syntax and semantics of composition policies and PESM diagrams so that a method for enabling plug-in of collaborations can be specified.

4. **Automatic PESM-based service composition.** At this point, our approach does not enable automatic composition, and it is left for further work to investigate whether a PESM diagram can automatically be generated.
Considered as a predefined, static graph, a PESM enables flexibility in choosing among its branches according to policies. By dynamically changing the set of policy rules associated with the PDPs one may dynamically change the branching criteria and thereby adapt to different situations. One may adapt to individual preferences by using profile information in the condition part of policy rules. One may even allow the set of rules that apply to a PESM to vary, depending on the different actors in order to achieve more flexibility. This would require that the policy rules are actor dependent, or local to each actor.

Allowing complete or partial PESMs to be dynamically downloaded and made available for execution can further increase the dynamic flexibility in two ways:

1. Substituting an existing PESM for a new one, by changing the reference to the PESM to use;

2. Adding new branches to an existing PESM by changing or adding policy rules (in its PDPs) to points to the new PESM.

Using these techniques one may provide extremely flexible configuration of actor behavior by starting from an initial PESM with just one PDP which is used to select its "application" PESMs according to its policies and preferences.

10 Conclusion

In this report we have elaborated our modelling approach to dynamic service composition by introducing the concept of a hierarchical PESM diagram. We have explained its use for a tele-consultation case. We provide an architectural framework for a policy-based dynamic composition system and we have explained how policy-enabled dynamic service composition can be deployed in the runtime system. A methodology is provided explaining how to use the policy-enabled concept to model, design and deploy dynamic composition of services. The report shows how the policy-based approach to dynamic composition functions at the implementation level. Policies are not just suitable for high level abstraction/specification mechanisms, but also are a part of the deployment in the runtime system. 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 distributed architecture described in Sect. 4.1 supports the separation of the policy-enforcement mechanism from the policies themselves, the monitoring function, and the services that are to be dynamically executed. As described in Sect. 4.1, the architecture provides a means of supporting the realization of policy-based dynamic execution of services. The objectives for the architectural given in Sect. 3.1 have been satisfied. We now consider the objectives for the methodology given in Sect. 3.2. In Sect. 5, we have defined a general methodology for policy-enabled dynamic composition of services. This methodology may be used with existing UML methodologies such as the rational unified
process (RUP) [10]. RUP describes which models should be created at each stage in the
development process. Similarly, the methodology presented here also describes which
models are created at the service, design and realization layers. The RUP development
phases involve a series of iterations. As explained in Sect. 5, each of the steps may be
iterated a number of times. The methodology describes how to use the models to spec-
ify, 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 secure services (dy-
amically). We believe that the methodology should be easy for a designer to understand
and use, however, we have not tested this by conducting experiments. This is first relevant
with tool support.

The concepts described here provide a starting point towards realizing policy-based
dynamic composition of services, in the runtime environment.

Acknowledgment

The research on which this reports has been partially funded by the Research Council of
Norway project SARDAS (152952/431). Thanks to members of the SARDAS project for
commenting on earlier versions of this report.
References


