A Policy-driven Approach to Dynamic Composition of Authentication and Authorization Patterns and Services

Judith E. Y. Rossebø
NTNU, Department of Telematics, Trondheim and Telenor R&I, Fornebu, Norway
Email: judith.rossebo@telenor.com

Rolv Bræk
NTNU, Department of Telematics, Trondheim, Norway
Email: rolv.braek@item.ntnu.no

***Abstract*— During the past decade, the telecommunication environment has evolved from single operator featuring voice services to multi-operator featuring a range of different types of services. Services are being provided today in a distributed manner in a connectionless environment requiring cooperation of several components and actors. This paper focuses on the incremental means to ensure access to services for authorized users only by composing authentication and authorization patterns and services. We propose a novel framework of authentication and authorization patterns for securing access to services for authorized users only, and we demonstrate how the patterns can be dynamically composed with services using a policy-driven approach.**

***Index Terms*— authentication, authorization, access control, policy, service composition**

I. Introduction

The evolution of service development in the telecommunications sector, driven by the success of the Internet, creates a demand for dynamic service development in order to continuously develop new services in a competitive market. There is a need for fast incremental development of services and applications, while maintaining availability properties.

We define a service as an identified partial functionality, provided by a system, component, or facility, to achieve desired end results (goals) for end users or other entities. The general notion of a service involves several service parts collaborating to provide the service to one or more service users. Authentication and authorization functionality needed to ensure availability is no exception and falls within this general definition of a service.

One of the core challenges of service engineering is to find practical ways to model services (partial functionalities) separately such that services may be composed into well functioning application systems satisfying availability requirements. This is especially challenging for services being provided in a distributed manner in a connectionless environment requiring cooperation of several components and actors (users).

If services were independent of each other, service composition would be quite straightforward. But services often depend on each other. Services also often depend on shared resources and service enablers. They may be provided to many interacting users, and users may have access to many services over the same terminals using shared resources and service enablers. This leads to the so-called crosscutting nature of services.

In [1] we outlined a two dimensional approach to service composition for which service and service parts depend on system components that are distributed across different computing environments. These domain entities such as users, user communities, terminals and resources are represented by agents in the system. We use the term agent in a general sense here to mean an entity representing and acting on behalf of other entities.

Service composition, in general, involves static composition at design time as well as dynamic linking and binding at runtime.

The new UML 2.0 collaboration concept [2] provides a structured way to define partial functionalities in terms of collaborating roles, and therefore it provides a promising basis for service modelling. It allows service parts to be modelled as collaboration roles, and service behavior to be specified using interactions, activity diagrams and state machines as explained in [3]. Moreover, it provides means to decompose/compose services using collaboration uses and to bind roles to classifiers defining system components. In this way, UML 2.0 collaborations directly support service modelling and service composition at design time. In addition they provide a framework to define so-called semantic interfaces as explained in [4] that can be utilized to ensure compatibility among interacting components both at design time and runtime.

The structure and linking of service components is to a large extent dynamic. Therefore, dynamic linking is a fundamental and general mechanism required in service-

This paper is based on "Towards a Framework of Authentication and Authorization Patterns for Ensuring Availability in Service Composition," by J. E. Y. Rossebø, and Rolv Bræk, which appeared in the Proceedings of the first International Conference on Availability, Reliability and Security (ARES), Vienna, Austria, April 2006. © 2006 IEEE.

The research on which this paper reports has been funded by the Research Council of Norway project SARDAS (152952/431).

oriented systems. Important mechanisms for service discovery, feature selection, compatibility validation, and access control can be associated with the creation and release of dynamic links. This linking may be seen as a process of dynamically binding roles to actors, taking the agent states and preferences into account.

Based on the conceptual model for service availability presented in [5], this paper focuses on the incremental means to ensure access to services for authorized users only by composing authentication and authorization (AA-) patterns and services. In order to address service availability, we see availability as a composite notion consisting of exclusivity, the ability to ensure access for authorized users only, and accessibility, the property of being on hand and useable when needed.

For fulfilling user expectations, availability depends more and more on the characteristics and requirements of the services themselves and the different requirements of certain users.

In [1] we have presented our work towards a framework and classification of authentication and authorization patterns. In this paper we provide a more thorough discussion of the specification of AA-patterns and the means to compose AA-patterns with services both statically and dynamically to restrict access to services to authorized users only. We explain in detail the policy driven approach to specifying composition of AA-patterns and services.

In summary, our contributions include: (1) a framework and classification of authentication and authorization patterns; (2) demonstrating that our framework can be applied to static and dynamic composition; and (3) showing that our framework can be used to specify and enforce policies governing composition of AA-patterns and services.

The rest of this paper is organized as follows: In Section II we state the requirements to the approach. In Section III we present our classification of authentication and authorization patterns. In Section IV we discuss our approach to specifying AA-patterns, and in Section V we discuss how we apply policies. Use of AA-patterns in static composition of services is addressed in Sect. VI, and use of AA-patterns in dynamic composition of services is addressed in Sect. VII. A discussion of related work is given in Section VIII followed by a summary and conclusion in Section IX.

II. Requirements to the approach

In order to explain the policy-driven approach, we formulate and motivate a set of requirements that the approach is designed to fulfill.

1) *The approach should facilitate specification of authentication and authorization patterns in a flexible and reusable manner.* It should be possible to model services independently and AA-patterns separately and then put them together and adapt them. As there are many different types of services, each with different authentication and authorization requirements, there is a need for a fine-grained approach that allows authentication and authorization design to be adapted to service requirements. The approach must allow for tailoring modifications with respect to e.g., strength of authentication provided and should be able to address the pitfalls that designers face in selecting and implementing authentication patterns to avoid faulty and weak implementations.
2) *The approach should be easy for a designer to understand and use.* Security requirements, such as availability requirements, are often not taken into account by developers in the design process for many reasons such as time to market and costs constraints, and lack of knowledge about security amongst designers and developers, as well as the complexity of the environment in which systems are deployed [6]. The approach should be understandable to the developer/designer and increase the designer's awareness of security issues while enabling the designer to address the issues systematically through choice and specialization of e.g. an authentication pattern.
3) *The approach should provide policy mechanisms that can be used for governing the binding of roles to agents in dynamic service composition.* This involves providing a means to specify constraints on the binding of roles to agents to ensure that service availability requirements can be achieved in a deployment. In order to ensure that service roles that are dynamically linked within a service execution are correctly linked, and to restrict which agents service roles can be bound to, there is a need for a means to define rules to govern the binding of roles to agents.
4) *The approach should provide a means for specifying the static composition of AA-patterns and services.* Static composition involves the assignment and composition of roles to form system components at design time. This involves building composite services using existing services and AA-patterns, but also defining roles and system parts so that they can collaborate with each other. The approach should provide a descriptive means to specify rules regarding ordering of collaborations in composition. In particular, requirements for which goals or states must have been achieved by collaborative parts before any other behavior is allowed to execute. The framework should also provide a means for modelling the dynamical restriction of the behavior involved in service composition, e.g. exceptions handling (such as in the case that authorizations are no longer valid, then the session should be forced to terminate).
5) *The approach should provide a means for supporting the dynamic composition of AA-patterns and services.* By dynamic composition, we mean both dynamic role-binding (i.e., creation and release of dynamic links) and dynamic composition of service role behaviors.

III. BACKGROUND

In this section, we briefly describe our classification of authentication and authorization patterns in order to provide the context for the discussion of our approach.

A. Authentication patterns

In [7], we present a flexible approach to specifying an authentication pattern as a specialization of one of a set of generic authentication patterns for use in service composition. Each pattern is modelled in UML 2.0 so that it may be re-used, but also may be easily adapted and adjusted depending on requirements such as security and performance requirements, for example regarding the the strength of the crypto involved related to the capacity available in the actual deployment.

Our classification is also motivated by the need to address behavioral considerations in the patterns. This means classification based first on the service provided, unilateral authentication or mutual authentication, then based on the number of messages involved in the pattern, e.g., one message for a one pass authentication protocol, two messages for a two pass authentication protocol. Each authentication pattern is modelled as a two-party collaboration, for which the behavior associated with the collaboration is modelled using a UML 2.0 interactions diagram. For the full classification, see [7].

In order to model authentication of one actor playing one role in a service collaboration to an actor playing another role in a service collaboration, we need a fine grained classification of authentication patterns. This is because the behavior required and strength of authentication required depends on the service to be deployed. In one case, unilateral one-pass authentication might be sufficient, e.g. for access to an online telephone catalogue. However, in another case, mutual two pass authentication may be required, as is the case for authenticating terminals and access points to each other in third generation mobile networks. Additionally, in the first example, a simple message containing a username and password satisfies the service requirements, whereas in the second example, hardware protected keys for use in a symmetric-based crypto protocol are required. The behavior required, the type of keying to be used, and the strength of crypto to be used is service dependent.

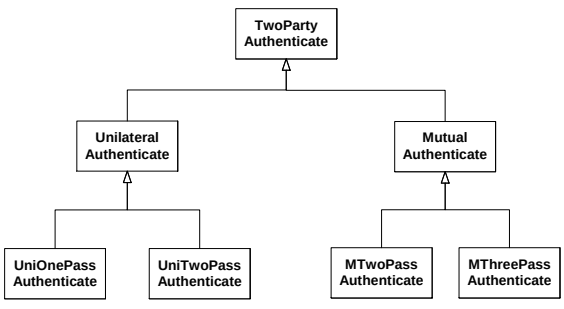

Figure 1. Authentication patterns

The aim is to make the developer more conscious in the choice of authentication technique to apply, while allowing flexibility with respect to the choice of protocol and algorithm(s) and other crypto techniques to be used. This allows the developer to focus on e.g. whether there is an issue such as timing regarding the number of messages involved e.g. one-pass, two-pass or three pass, or should symmetric or asymmetric keying be used, before choosing the protocol and algorithm in the instantiation of the pattern. Fig. 1 shows our classification of authentication patterns.

A generic two party authentication pattern involves communication between the two parties to establish the identity of one of the parties in the case of unilateral authentication, or both in the case of mutual authentication. Messages are generated and exchanged between the parties, at least one message/pass is required for unilateral authentication, and at least two messages/passes are required for mutual authentication. These are generic patterns that do not bind a particular protocol or algorithm. Once a generic pattern is selected, the authentication pattern can be further differentiated in specializing the pattern depending on the type of keying, e.g., symmetric or asymmetric, to be used.

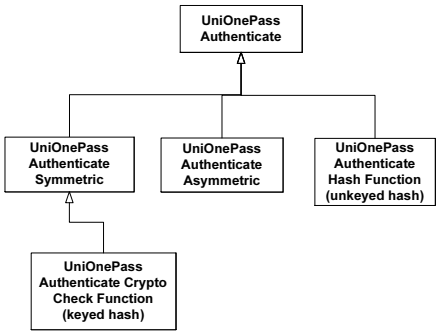

Figure 2. Unilateral one pass authentication patterns

For example, the unilateral one pass authentication pattern may be specialized as illustrated in the UML 2.0 class diagram shown in Fig. 2. There is a class for all unilateral one pass patterns employing symmetric crypto techniques, that is, for which the authenticating party and the party requesting authentication share a common secret key, which is used in the crypto protocol. Similarly, there is a class for all unilateral one pass patterns employing asymmetric crypto techniques, and a class for all patterns for which the unilateral one pass authentication algorithm employs a Hash function.

The patterns are then further specialized with respect to the authentication technique, or cryptographic protocol and algorithm(s) to be applied, e.g., for the unilateral two-pass authentication pattern, the HTTP digest authentication protocol with the MD5 hash algorithm may be applied [8]. By doing this, we separate out the choices that must be made by the developer, and pinpoint each of the levels of specialization for awareness. This is because it is not enough to choose a general model and apply just any technique or protocol and assume that required level

of security is achieved. There are altogether too many examples illustrating that depending on choices at each of these layers, the actual implementation can be flawed.

One example of this is the Microsoft challenge/reply handshake protocol, used in Microsoft's Point-to-Point Tunnelling Protocol (PPTP). In this example, a design flaw in the protocol and a choice of a weak password hashing algorithm both contributed to the reported weakness of the authentication implementation [9]. Additionally, there were other flaws in the implementation itself. It is because flaws may be introduced at different stages in authentication design and implementation that we have chosen to classify patterns separating stages of specialization. These stages are as follows: First, a general pattern is chosen from the classification in Fig. 1. Then, the pattern is specialized according to technique, e.g., if crypto is to be employed, then a choice must be made between symmetric or asymmetric keying, and then a protocol must be chosen along with algorithms or functions required by the protocol. If desired, an original protocol and algorithm may be designed for the application and specified during the design process. This allows the developer to analyze the strength of the authentication at each stage of specialization of the models, so that flaws and weaknesses may be discovered and corrected.

It is important to distinguish between weak versus strong authentication, and weaknesses and errors that arise simply due to implementation errors. The strength of the authentication pattern can be tuned with respect to the combination of the protocol, the algorithm and the key-length. However, errors in implementation can significantly weaken the authentication mechanism delivered. Assurance techniques such as e.g. use of the Common Criteria [10] may help in the latter.

B. Authorization patterns

In order to describe any authorization pattern, it is important to recognize that any authorization pattern requires that authentication has been performed before any authorizations may be granted. Authentication and authorization patterns are combined to describe how access rights are granted and are thus essential to access control. Additionally, an access control model is required for access rights administration.

There are two basic authentication and authorization architectures [11]:

User Pull: Authentication is performed by an access server, which also issues authorizations to the user. The user then presents authorizations directly to the service.

Server Pull: The service centralizes information about user entity authorizations on an access server. The service authenticates the user. When the user attempts to access the service, the service queries the access server to determine whether the user is authorized.

These architectures provide a means for handling authorizations in a centralized manner. For the full classification of these architectures as patterns, see [7].

IV. Specification of AA-patterns

A. Using UML 2.0 collaborations

A UML 2.0 collaboration diagram for the generic two party authentication pattern is given in Figure 3. The collaboration diagram shows that the `authenticatee` role cooperates with the `authenticator` role.

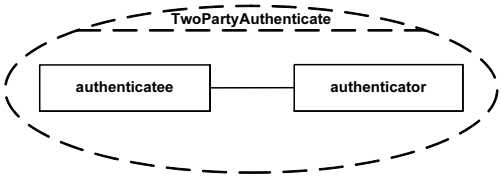

Figure 3. Collaboration diagram for the two party authentication pattern

A detailed view of a specialization of this pattern for unilateral two pass authentication is shown in Fig. 4. This view expresses more completely the properties that the system components (such as agents) must have in order to successfully participate in the pattern. Any instance playing the `authenticatee` role must possess the properties specified by `responder` and any instance playing the `authenticator` role must possess the properties specified by `challenger`. The instance playing the `authenticatee` role must possess a secret, and the instance playing the `authenticator` role must possess knowledge that is mathematically related to the secret. The instance playing the `authenticator` role must be able to generate a `challenge`, which is sent to the instance playing the `authenticatee` role, and validate the `response`. Similarly, the instance playing the `authenticatee` role must be able to generate a `response` to the `challenge`. The constraints (on the properties that the instances playing the roles must possess) are declared as invariants and pre-conditions using the object constraint language (OCL).

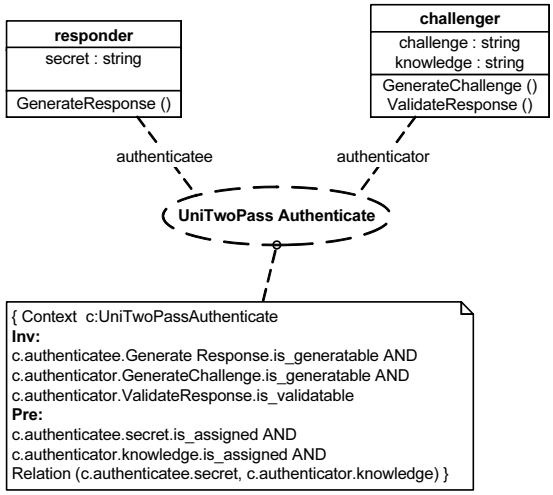

Figure 4. UML 2.0 Collaboration diagram for unilateral two-pass authentication, detailed view

Three invariants are declared: The first and third invariants are used to check that the instance playing the

`authenticator` role is deployed on a part of the system (terminal/node) with the required processing and computing capacity required to generate the challenge and to validate the response. Similarly, the second invariant is used to check that the instance playing the `authenticatee` role is deployed on a part of the system (terminal/node) with the required processing and computing capacity required to generate the response. The reason for declaring these invariants is to ensure that the protocol and algorithm chosen are not too processor intensive for the parts on which they are deployed so that the authentication protocol can run whenever the collaboration is instantiated. The motivation for this is to ensure that service requirements regarding accessibility [5] are fulfilled when this authentication pattern is composed with service components/parts.

The two pre-conditions check that `secret` and `knowledge`, respectively, are assigned before the collaboration can instantiate. The third pre-condition checks that there is a mathematical relationship between `secret` and `knowledge`. This means that a check can be performed to ensure that there is a pre-existing mathematical relationship between `secret` and `knowledge` as required by the authentication pattern to be deployed. The OCL pre-conditions can be used to perform a boolean check to confirm that the *a priori* conditions for the authentication protocol are fulfilled. This formalization of the mathematical relationship between `secret` and `knowledge` has been chosen in order to be general enough to allow for alternative crypto protocols to be specified at later stages in development. Note that if symmetric keying is used, then `secret` = `knowledge`.

We model the User Pull authentication and authorization services as a UML 2.0 collaboration that defines three collaborating participants that interact to implement the User Pull authentication and authorization behavior: these are the `User`, `Access Server`, and `Service Access Filter` roles. Application of certain AA-patterns to the User Pull services is represented by three collaboration uses as illustrated in Fig. 5.

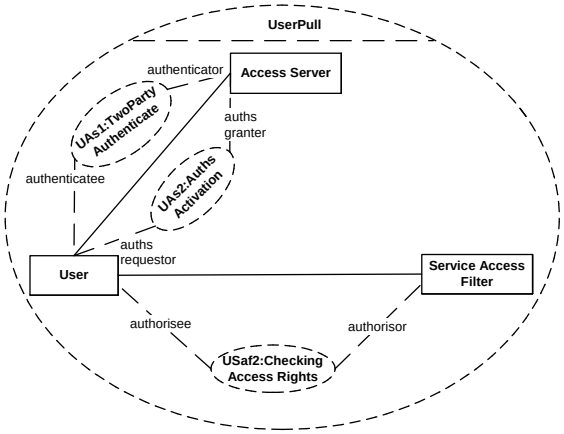

Figure 5. User Pull patterns

Although not shown in the authentication and authorization patterns presented above, an access control model is needed to administer access rights (permissions) and enforce access control policies. Several models for access control have evolved such as discretionary access control (DAC), mandatory access control (MAC), and others [11]. A detailed overview of different access control models is given in [12]. Role-Based Access Control (RBAC) has emerged as a scalable alternative, and has been the focus area for recent research on access control resulting in numerous model variants. In this report, we assume that a RBAC model is used with the AA-patterns.

RBAC-role activation rules, and authorization rules are administered by the RBAC infrastructure, and distributed to the AA-patterns and services. Therefore, there must be an interface from the `Access Server` towards an RBAC infrastructure. For an RBAC model and an approach to modelling RBAC policies using UML, see [13]. Access control policies are enforced based on RBAC-role activation rules and authorization rules. RBAC-activation rules are used to manage and activate RBAC-roles acquired by the agent. For example, a service role may or may not be allowed to be played by an agent depending on the RBAC-roles acquired by the agent.

B. Using semantic interfaces

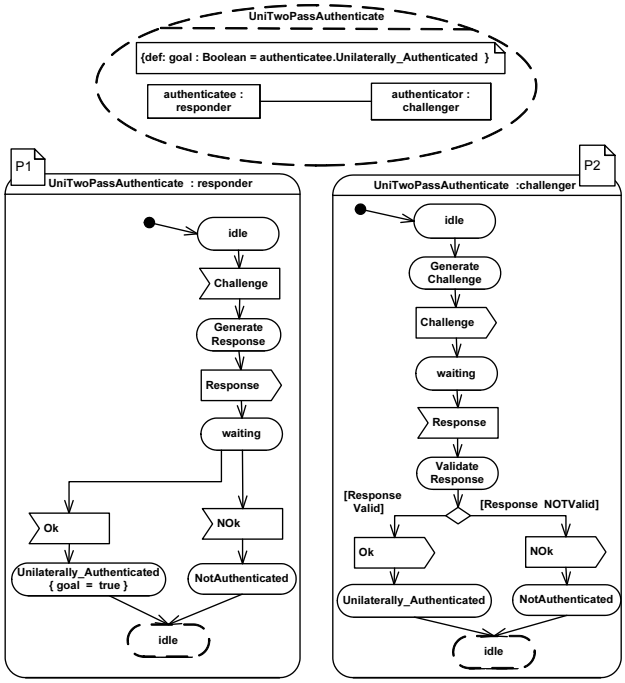

Figure 6. Semantic interface for UniTwoPass Authenticate

A *semantic interface* is defined as a collaboration with two state transition diagrams defining the interface behavior for each of the roles involved in the collaboration and possible expressions stating the goals of the collaboration. In [4] it is described how semantic interfaces can be defined based on role modelling and simple goal expressions. The focus is on checking the compatibility of different service components involved

in the provisioning of a service. Definition of semantic interfaces allows us to validate the interface behavior rather than validating the complete component behavior. Semantic interfaces facilitate validation of both safety and liveness properties. It is pointed out in [4] that UML 2.0 protocol statemachines are not sufficient, and the authors propose a form of UML 2.0 state machines for two way interface behavior as shown in Fig. 6.

In this figure, the UML 2.0 collaboration for unilateral two-pass authentication pattern is shown with two state machines that define the visible behavior of the two roles participating in the pattern. In addition to syntactical interfaces, semantic interfaces define the visible interface behavior and goals of the collaboration. In this case, the semantic interface defines the interface behavior and goals of the `authenticatee` and `authenticator` roles.

V. AA-PATTERNS AND POLICY

In [14], a policy is defined as information which can be used to modify the behavior of a system. This definition of policy covers as such role-binding constraints as well as user preferences, but also constraints on the triggering of behavior between components.

During service execution, dynamic role-binding provides a means for governing service execution as outlined in [15], using a policy-driven approach to control invocation of service roles. Our classification and approach to specification of policies is also motivated by [16] and by [17].

In our policy driven approach to composing AA-patterns with services, we are concerned with defining selective mechanisms for enabling the joint behavior of objects rather than one object individually. As such, policies should make it possible to provide information on sequencing of collaborative behavior as well as the triggering of collaborative behavior when policy constraints are fulfilled. It should be possible to provide information on the ordering of service goals as well as the relationship between collaboration uses composed to provide services. One way of doing this is expressing a composition policy as a UML 2.0 dependency between two collaboration uses involved in a composed service.

Our notion of a role-binding policy specifies requirement/objectives specifically for the instance playing a certain role in the collaboration. This includes e.g. constraints the role imposes on any agent it may be bound to as well as conditions an agent may pose regarding which roles may be bound to that agent depending on agent states and preferences. Role-binding policies typically consist of context dependent constraints. In the context of authentication patterns, a collaboration policy is as such a requirement/objective for the collaborative behavior of the authentication pattern as a whole, whereas the role-binding polices are defined specifically for each of the two collaboration roles, `authenticatee` and `authenticator`.

Role-binding policies associated with a role may consist of:

- Role requirements, e.g. on which properties the instance (agent) playing the role must have in order to successfully participate in the collaboration. For example, for the unilateral two pass authentication pattern shown in Fig. 4, any instance playing the `authenticatee` role must possess a `secret`.
- Agent requirements, which may specify constraints on what the agent playing the role is allowed to do or which agents are allowed to play the role, e.g., only a `UserAgent` is allowed to play the `authenticatee` role. The constraint may specify requirements that the agent must satisfy in order to play the role, e.g., in order for the collaboration to be successful with respect to service availability requirements.
- Deployment requirements, e.g., requirements for the platform that the role is deployed on in order for the collaboration to be successful with respect to service availability requirements. For example, an instance playing the `authenticatee` role must be able to generate a `response`. This means that agent playing the role must be deployed on a part of the physical system with the required capacity available.

A collaboration role participating in an AA-pattern may have requirements on what the agent must be able to support in order to play a role. We therefore need to determine that the agent has the properties/characteristics required in order to play the role, such as support for a specific algorithm. If it is determined that the algorithm to be used is not supported, it may also be possible to download this (as a sort of extension to the role play) to the agent allowing for the role to be played anyway.

A role-binding policy held by an agent defines conditions and constraints on which roles can be played by the agent and defines rules in terms of:

- Pre-conditions for invoking a role such as conditions on the other agent involved in the collaboration or conditions on which roles shall have been performed (e.g. AA-roles).
- Preferences of the agent, such as types or multiplicities of roles that can be bound to the agent.
- Deployment conditions. This provides e.g., information about the resources available. This may include information about the type of terminal/node/user equipment that the agent is deployed on, e.g., the terminal is a 3G telephone with a smartcard, information about the operating System/ or software supported, and other contextual parameters. Essentially, providing information about which service availability constraints can be met by the platform the agent is deployed on and which influence whether or not a role can be played by an agent.

For example, the policy held by a user agent `A` may state that user agent `A` is only allowed to participate in a `voicecall` service, playing the `callee` role if the user agent playing the `caller` role has been authenticated, authorized, and identified, and the invitation is received between 6 PM and 11 PM.

A. Specifying policies

Most of the work in the literature on defining policies focuses on organizational policies e.g. RBAC policies and Role Based Management (RBM) policies in which a role is an organizational concept representing the specification of the behavior associated with a particular position in the organizational context [18]. Although policies for governing service execution are addressed in [15], specification of rules for defining such policies is not addressed. An architecture for policy definition and call control policies is given in [19] and provides some high level ideas for defining policies for use in enhancing and controlling features in the context of call control in telecommunication services. In this section, we refine some of these ideas and we provide our approach to defining policies.

Definition: A policy is a rule of the following form: If *condition* C and *trigger* T then *action* A and *goal* B.

- *The condition part* defines constraints on its applicability. The constraints are predicates which restrict role behavior in service composition. We may specify constraints as invariants, and pre-conditions in OCL, or more specifically, in Ponder [20].
- *The trigger part* describes when the policy should be applied. The trigger is the event that e.g. invokes the execution of a collaboration subject to the constraints stated in the condition part. The trigger part of a policy for governing service invocation is important for achieving dynamic linking in service composition. The trigger is specified as a message in UML 2.0, e.g., a signal or call.
- *The action part* defines what is to be done when the trigger event has been sent given that the constraints stated in the condition part hold. Examples of actions are: *bind role r to agent A*, and *execute collaboration C*.
- *The goal part* defines what is the desired result when the policy is applied. These goals may be specified as post-conditions in OCL.

Although a trigger part is not specified in policy rules in general, e.g. in [16], the trigger part is essential for applying a policy approach to service engineering. For a role-binding policy, the trigger establishes when the policy applies, e.g., when the role request message is sent. Specifying composition policies allows us to make relationships between collaboration uses explicit as well as providing a means for sequencing service goals. e.g., a composition policy may state that the goal `unilaterally_authenticated` must be achieved before the goal `auths_activated` can be achieved.

B. Applying policies to service composition

The following outlines how policy is applied in our modelling:

1) The role-binding and composition policies are specified using e.g., OCL. Policy conditions are stated as invariants, and pre-conditions. Triggers and policy actions are stated as UML operations, and goals as post-conditions. For example, if the invariants and pre-conditions stated in the role-binding policy are satisfied, then the instance can play the role. Another example is use of composition policies to demonstrate dynamic linking of collaboration uses. The composition policy is declared as a UML 2.0 dependency of type $\ll policy \gg$. For example, the composition policy may declare that a pre-condition for the execution of one collaboration may be that the instantiation of another collaboration has reached a certain goal. These policies are specified at design time.
2) At design time static checks are performed on e.g., the projection from an actor's state machine to the semantic interface. Checks are performed on role compliance. This includes checking that the actor satisfies the conditions and properties given in the role-binding policy. This implies that the actor is typed with the interface.
3) At run-time, policy controls are performed on the interfaces, dynamically. At run-time it is enough to check that both instances are of the type that is required on the semantic interface. Whether collaboration policy is satisfied is checked, as well as checking e.g., whether access control policy rules are satisfied.
4) Access control policy enforcement is performed dynamically in an instantiation of the `Checking Access Rights` collaboration by the instance playing the `authorisor` role.

We have found that it is useful to declare the role binding policies in the semantic interfaces for use in validation that the security properties are preserved in composition of the pattern with services.

VI. Composing AA-patterns and services statically

As discussed in [7], AA-patterns behavior may be invoked in two different situations:

When creating a new session, by performing a role request and performing dynamic role binding. This requires general mechanisms to ensure that the role is invoked only if authentication and authorization policies are satisfied. If role `r` is requested, and a policy specifies that authentication and authorization is performed first, then the necessary AA-behavior must be performed first and a desired goal must be reached before the service is invoked. In this case an AA-goal is a precondition for the service invocation.

During session behavior, this is required when the session and its roles contains features or accesses objects that demand fine-grained, dynamic authentication and/or authorization. This case is trickier because it requires a tighter integration of service behavior and AA-behavior. In our work, we model this using service access filters, and policies, e.g. restricting role behavior. This entails adding screening behavior that filters out unauthorized operations. It also requires that it may be possible to force

termination of a session if authorizations are no longer valid. We have currently modelled this as an `Interrupt` collaboration. Another approach is to invoke a restricted role behavior only capable of doing authorized operations. Applying the appropriate role-based access control model for issuing authorizations, and checking authorizations upon accessing a particular service or object makes such fine-grained, dynamic authorization possible.

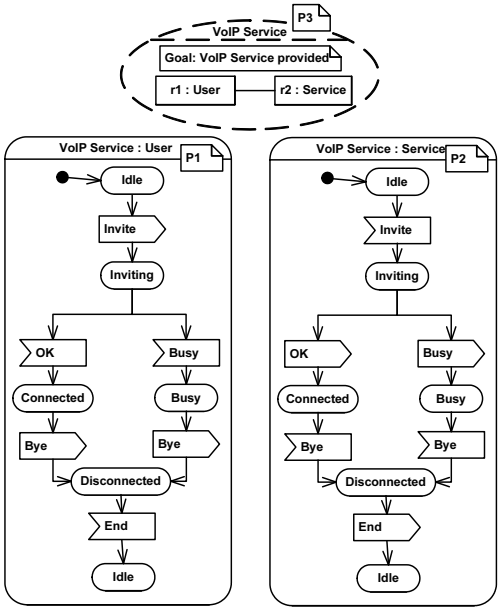

Figure 7. VoIP service defined as a semantic interface

Let us assume a voice over IP (VoIP) service, `VoIP Service`, defined as a semantic interface with roles `r1` and `r2` as shown in Fig. 7. We model the view showing the user to VoIP service provider only, to keep the example simple. Further assume that agent `A` requests a session of `VoIP Service`, and role `r2` from agent `B`.

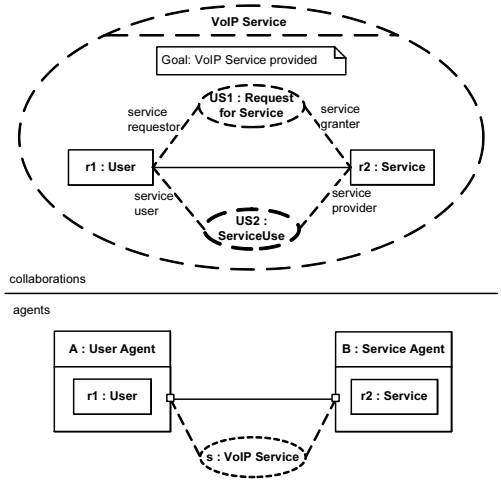

Figure 8. VoIP Service: binding roles to agents in service composition

The collaboration `VoIP Service` may have a collaboration policy P3 specifying that the agents playing `r1` and `r2`, in our case agents `A` and `B`, shall be different agents. The agents may specify conditions that govern which roles can be played by the agent. Agent `B` may, for instance, specify that a precondition for invoking `r2` is that agent `A` is authenticated and authorized e.g. applying `Userpull`. Similarly, agent `A` may specify that a precondition for invoking `r1` is that agent `B` is authenticated and authorized. It is natural to express these conditions as part of the role-binding policies, using OCL. If the AA-properties have not been established yet then, it

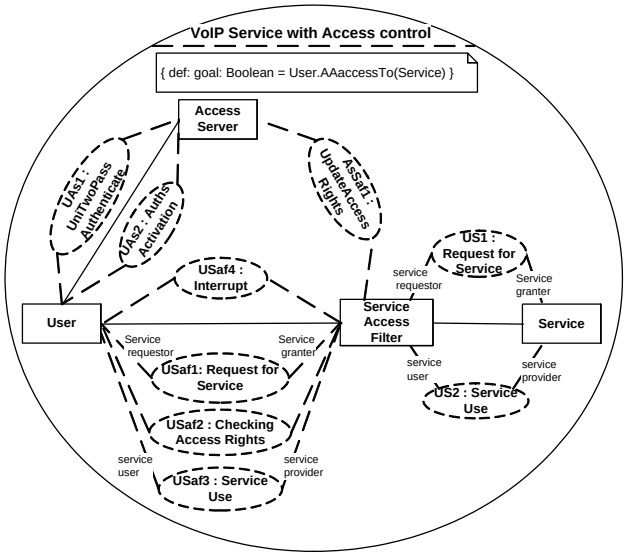

Figure 9. VoIP Service composed with User Pull patterns

is necessary to invoke AA-services resulting in the desired AA-properties before invoking `VoIP Service`. In the most general case agent A and agent B must negotiate and agree on the AA-patterns to apply. In many cases agent B may select the patterns and return the decision to agent `A`. Then the AA-services are performed and only if successful, is the requested `VoIP Service` invoked.

In order to demonstrate composition of VoIP Service with AA-patterns, we decompose `VoIP Service` as shown in Fig. 8. The collaboration use `Request for Service` represents the initial request for use of the VoIP service by the instance playing the `User` role. There are several alternatives to determine what is a result of this initial request. One option is that a service manager is implemented in the system, which in response to the request from the user determines that authentication and authorization is required for access to the service. A set of AA-patterns is then selected for composition with the service. Another alternative is that the instance playing the `Service` role determines which AA-patterns are needed and that instances of `Access Server` and `Service Access Filter` are required to perform authentication and authorization. If these are successful, then the requested `VoIP Service` is invoked.

As this article focuses primarily on modelling techniques/alternatives for enabling static composition of AA-patterns and services at design time, we do not discuss

the dynamic linking that occurs from Fig. 8 to Fig. 9. We assume, therefore that the decision to compose the AA-patterns with the VoIP Service as shown in the collaboration `VoIP Service with Access control` has been made. We now discuss the different modelling techniques/alternatives for achieving static composition.

In Fig. 9 we demonstrate static composition of `VoIP Service` with the User Pull authentication and authorization patterns. This involves re-use of the two collaborations: `Request for Service` and `ServiceUse`. The re-use of these two patterns is needed in order to enable the instance playing the `ServiceAccessFilter` role to act as a proxy between the instance playing the `User` role, and the instance playing the `Service` role. This enables the instance playing the `Service` role to require authentication and authorization before allowing a user to access the service. The `VoIP Service` session may require additional, fine grained authentication, and authorization checks, however. This calls for screening or other mechanisms during service execution, unless it is possible to constrain the service that is invoked to what is permitted. The instance playing the `Service` role, may require that these additional, fine grained authentication, and authorization checks are performed by the instance playing the `ServiceAccessFilter` role. We model these as the following collaboration uses: `UpdateAccessRights`, for updating the status of the user authorizations, and `Interrupt`, for terminating a service session if user authorizations are no longer valid.

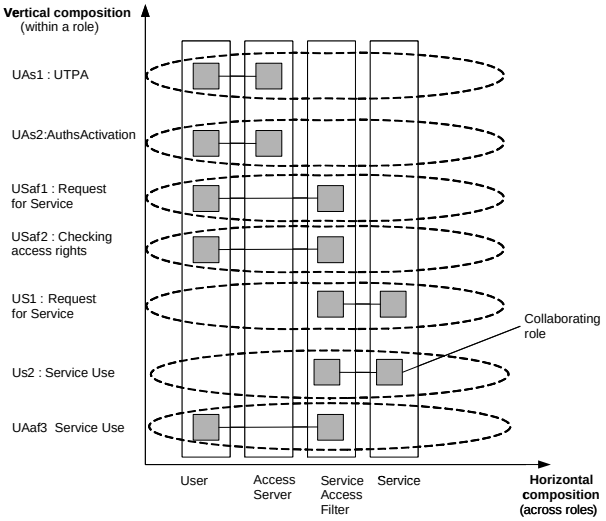

Figure 10. Diagram showing composition of collaboration uses, ordered in top down sequence

AA-pattern collaborations describe reusable elements. During instantiation of a collaboration, various checks are needed to ensure that the participating agents can satisfy requirements, conditions and properties, stated in policies. Role-binding policies are used to check the compatibility of the role with the agent playing the role. When binding roles, the semantic interface between two roles is also bound, that is, the roles must also be compatible on a

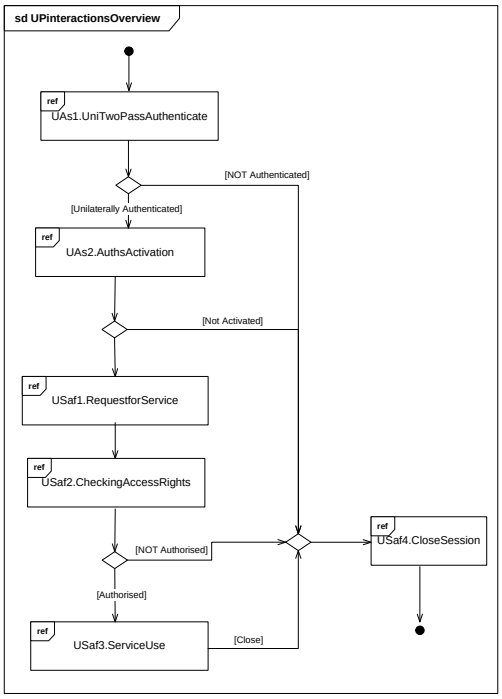

Figure 11. Interactions overview diagram for composing sequence diagrams

semantic interface with each other. Work on validating the compatibility of roles and consistency checking to ensure the correctness of roles has been done by [21], [22], and [23].

Fig. 9 gives a graphical overview and provides a decomposition into interfaces that are quite modular and reusable. However, the overall coordination (referred to as choreography in the SOA context) is not evident. In addition to providing information about the static structure, we also need to provide information about the ordering of the associated behavior. Fig. 10 shows how roles in the different collaborations are composed and shows how these are ordered in a successful service execution.

As explained, above Section IV, for each of the two party collaborations we model the behavior associated with the collaboration using semantic interfaces and goals. In addition, a UML 2.0 interactions diagram corresponding to the semantic interface may be designed for each collaboration. These can then be referred to in a UML 2.0 interactions overview diagram such as in Fig.11.

There are several alternatives for modelling the sequencing of the behavior associated with the collaboration uses shown. One alternative is use of an interactions overview diagram as shown in Fig. 11. Such interaction overview diagrams are not entirely suitable for expressing interrupting and disabling such as the termination of a user session if authorizations are no longer valid. To model such dynamic exceptions, a UML 2.0 activity diagram may be useful, modelling the dynamic exception using an interruptible activity region [2].

Modelling the `Interrupt` collaboration behavior in service composition is not easy, as it constitutes an excep-

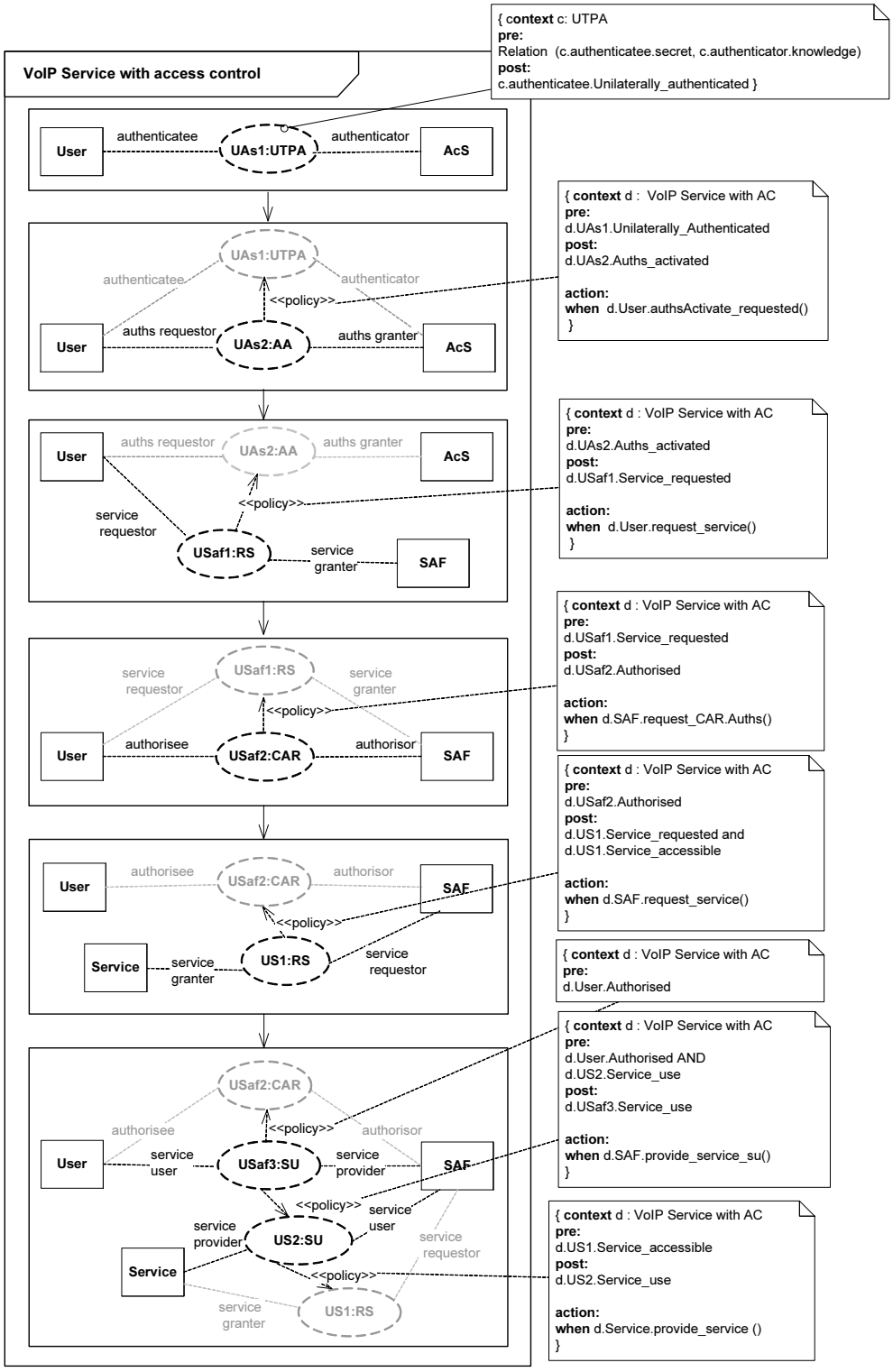

Figure 12. Goal sequences with policies as UML 2.0 dependencies.

tion behavior. Indeed, the `Interrupt` collaboration is an example of a forced feature interaction. Halvorsen and Haugen have presented a method for handling exception in sequence diagrams in [24].

A goal sequence [3] provides supplementary information to a collaboration diagram. While a collaboration such as given in Fig. 9 provides static structural information about the roles and collaboration uses involved in a composition of re-usable units such as AA-patterns and services, a goal sequence provides additional information about the ordering of dynamic behavior associated with the collaboration uses. Fig. 12 models the positive sequence of behavior associated with the collaborations in order to achieve authenticated and authorized access to a

service.

We have extended the idea of a goal sequence given in [3] to include modelling composition policies as UML 2.0 dependencies with keyword <<policy >> as illustrated in Fig. 12. This allows us to express constraints on the ordering of the behavior associated with collaboration uses and may also allow us to express policies governing dynamic interupt exceptions. The composition policies, modelled as UML 2.0 dependencies with keyword <<policy >> allow us to specify conditions that must be true in order for the behavior associated with a collaboration use to execute. Each instance of <<policy >> is annotated with the policy specified in OCL.

The policies declared provide additional information on conditions required for the behavior to run correctly and according to availability requirements. The post-conditions declare goals that have been defined at design time in the semantic interfaces for the collaborations involved. For example, the goal `unilaterally_authenticated` is declared in the semanic interface for `UTPA` in Fig. 14.

As shown in Fig. 12, a collaboration policy is declared using OCL for the first collaboration use in the sequence, the instance `UAs1` of `UTPA`. This collaboration policy is declared at design time, when the `UTPA` collaboration is designed, along with UML 2.0 interactions and the semantic interface. The goal for the collaboration, `c.authenticatee.Unilaterally_authenticated`, is also declared in the collaboration policy as an OCL post-condition. The reaching of this goal, becomes a pre-condition in the composition policy declaring when the behavior associated with the instance `UAs2` of `AA` can execute.

We prefer to model the composition policies as UML 2.0 dependencies in a goal sequence as apposed to declaring such policy dependencies in a UML 2.0 collaboration overview such as the overview shown in Fig. 9. This is because, the dependencies would cross over several collaboration uses, and often cross each other, making the result very difficult to read and understand. By using a goal sequence instead, the policies can be expressed clearly, sequentially, and dynamically.

A. Steps for composing AA-patterns with services

We sum up our approach to composing AA-patterns at design time in the following steps:

1) Determine which AA-patterns should be applied. In this step, it is determined based on service availability requirements, which set of AA-patterns will be applied. This involves deciding whether AA-behavior should be applied separately for each service in parallel, or whether some form of centralized authentication and authorizations can be used, requiring that `UserPull` or `ServerPull` should be applied. The decision to apply `UserPull` or `ServerPull` involves deciding whether user authorizations will be stored on a centralized access server, and presented by the access server to the service, or whether authorizations will be distributed to the user and presented by the user to the service. Regarding choice of authentication pattern to apply, we discuss this in more detail in [7].
2) Decide whether sequential invocation at the beginning of a session only is sufficient, or whether more fine grained control during session behavior is also required.
3) Once the set of patterns to apply has been chosen, specifications/models for each of the AA-patterns and the service to be composed are designed. These are: UML 2.0 collaborations annotated with goals and a collaboration policy, semantic interfaces annotated with role binding policies for each of the two participating roles, and UML 2.0 interations. Declaring role-binding policies in the semantic interface for each of the two collaborating roles involved will enable us to validate that the required conditions and requirements have been fulfilled when composing the pattern with other AA-patterns and services. The semantic interface may also be annotated with the collaboration policy and goals.
4) Specification of the collaboration showing composition of AA-patterns with the service, annotated with the collaboration policies. To supplement this collaboration overview diagram, a goal sequence diagram is also provided, e.g. as shown in Fig. 12.
5) Consistency checking of the model in the previous step using semantic interfaces. Consistency checks related to goals, and to role binding policies. In this step we will also evaluate whether or not availability properties are preserved under composition.

While these steps address static composition at design time, it is also possible that agents representing users in the system, negotiate on behalf of end-users, service providers and system resources to achieve dynamic composition at run time, as we discuss in the following section.

VII. Dynamic Role-Binding Using Semantic Interfaces

We define the semantic interfaces (SI) separately and validate (model check) each SI type separately to ensure safety and liveness properties. In this sense, a semantic interface is a type that may be used at design time to ensure the correctness of (static) associations and at runtime (as meta information) to ensure the correctness of (dynamic) links.

Role binding policies declare requirements for the classes and instances a role may be bound to. Actor/role types are then designed for the runtime system and are model checked against the SI to validate that the interface behavior required by the collaboration (e.g., UTPA) is satisfied.

In the meta data for the runtime system, this information is stored in files in the database as part of the management system, and forms part of the data model

for the runtime system. In the meta data, for example, we know that the instance can play the `authenticatee` role of type `responder` in an instantiation of the `UTPA` collaboration and be able to satisfy requirements regarding strength of authentication provided, and response times involved in the exchange.

At runtime, dynamic role-binding is performed using the actor and SI type information to ensure compatibility of dynamic links thereby guaranteeing that the links satisfy the properties of the SI.

A. Example

The SI for a specialization of the unilateral two pass authentication pattern, using the HTTP digest protocol with the MD5 algorithm [8], is given in Fig. 14 along with the `UserAgent` and `Service Agent`, which represent user and service domain entities and resources, respectively. In order for an instance of `responder` to be bound to the `UserAgent`, the role-binding policy `P1` must be satisfied. Similarly, for binding and instance of `challenger` to the `Service Agent`, the role-binding policy `P2` must be satisfied. The role-binding policies have been specified in OCL for the roles involved in the two party authentication pattern. For example, the OCL Boolean constraint `is_generatable` is declared to address performance aspects of the authentication exchange. The aim is to ensure that the system resource on which the role/agent is deployed is able to perform the operations involved in the authentication exchange within QoS requirements.

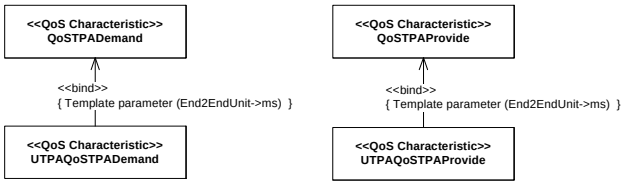

Figure 13. UTPA QoS class definition

The OCL Boolean constraint `is_generatable` has been defined using [25], and `is_generatable` evaluates to true means that the *required QoS* demanded by the role in order to satisfy accessibility constraints is met by the *offered QoS* of the resource in the deployment model. In order to represent the quality values we need to define `is_generatable` for use in dynamic role-binding, we define a simplified quality model, as shown in Fig. 13 based on the Quality Model given in Annex B of [25]. In this case, we resolve all temporal units with the unit milliseconds (ms). This simple model may be expanded and refined with additional characteristics.

The OCL constraint `is_generatable` is a Boolean check that is defined is OCL as follows:

```
{ Context UTPA
if UTPAQoSTPAProvide ≤ UTPAQoSTPADemand
then self.authenticatee.generateMD5response.is_generatable = true
else self.authenticatee.generateMD5response.is_generatable = false
endif
}
```

The required QoS defines the maximum allowed time to generate the MD5 response, and is specified on the SI type annotated to the statechart for the `responder` role. In this case, the required QoS is the worst case for generating the MD5 response is 10 ms. Similarly, the deployment model for the agents provides information about the offered QoS of the resources. The offered QoS by the resource is 10 ms or better for the agent that is to be validated against the required QoS. For this case, as shown in Fig. 14, `is_generatable` evaluates to true.

Support for java-based role-binding, and collaboration policies has been implemented in ServiceFrame [26]. Services can be specified by both end-users and service providers to handle availability properties. Extensions of ServiceFrame for validation interface behavior by checking consistency are also being investigate by the students at the Norwegian University of Science and Technology (NTNU). Extensions for modelling collaborations and deriving interface behavior associated with these have also been implemented. Work is ongoing regarding consistency of service roles using semantic interfaces.

VIII. Related Work

Yoder and Barcalow [27] were the first to apply design patterns to the security domain presenting the Single Access Point Pattern in [27]. In [28], patterns for authorization and access control are addressed. Brown, Divietri, Villegas, and Fernandez have documented a high level design pattern for authentication of clients to a server [29]. Consistent with our approach, the pattern allows for the implementation of different authentication methods such as password-based, challenge response, or multiple challenge response. However, our approach to designing patterns allows for application of the authentication pattern to the peer-to-peer environment as well. Additionally, we provide a means to specify more details at later stages of development depending on the requirements of the authentication protocol and algorithm. In [30] Fernandez and Warrier provide an authorization pattern, integrated with a variant of the authenticator pattern. This authorizer pattern is actually an application of Yoder and Barcalows single-point-of-check pattern [27], and is also an example of a server pull authentication and authorization architecture. Although these and other different authors have addressed authentication patterns and authorization patterns separately, we are not aware that a framework addressing authentication and authorization patterns exists. To our knowledge, application of such a framework to service composition is also a new approach.

IX. Conclusion

We have presented a framework of authentication and authorization patterns together with a policy-driven approach to composing services and AA-patterns to restrict access to services to authorized users only. This involves specification of the AA-patterns using UML 2.0 collaborations and semantic interfaces annotated with policies specified using OCL. We have demonstrated that our

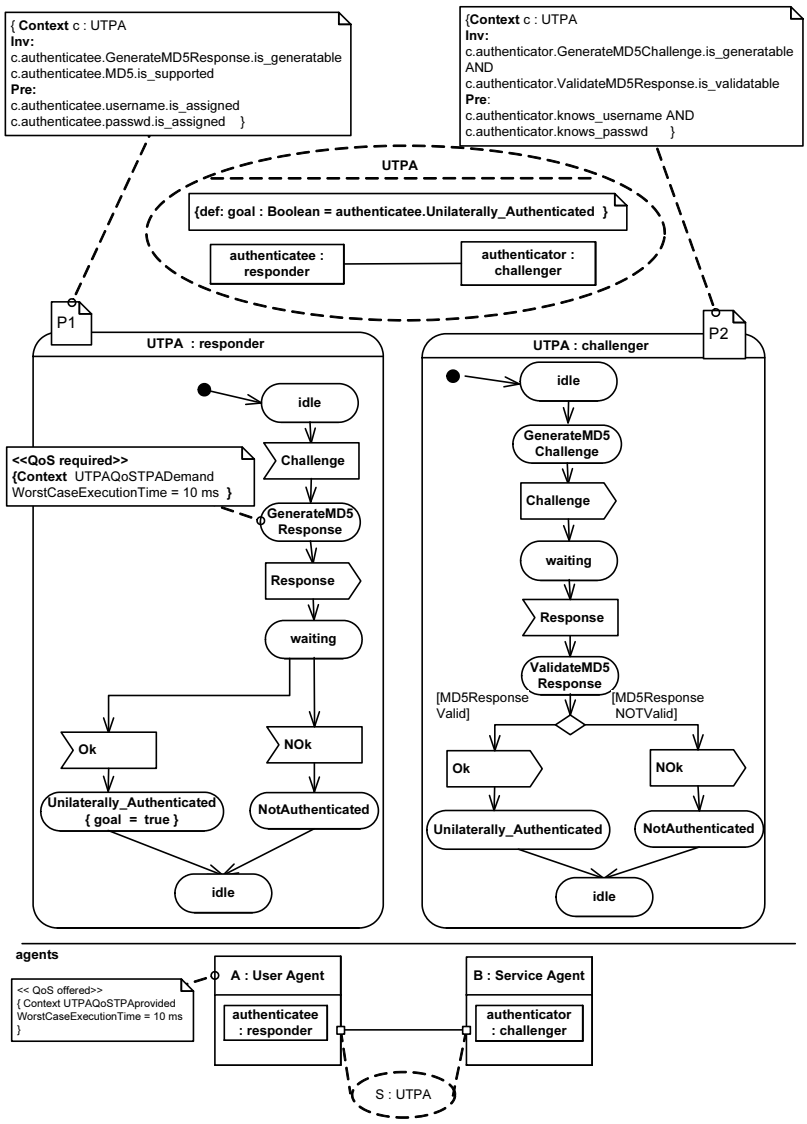

Figure 14. Unilateral two pass authentication defined as a semantic interface

framework can be applied to static and dynamic composition of services. Furthermore, we have demonstrated how the specifications may be annotated with role-binding policies, collaboration policies, and composition policies to enable us to validate that required conditions and availability properties hold when composing AA-patterns with services.

This policy-driven approach is useful for application to service composition because there are significant differences between different authentication techniques that must be modelled for use in service composition, depending on the service collaboration roles and service behavior involved as well as differences in the resources available in the deployment platform. This validates the need for a finer-grained classification of authentication patterns as discussed above in Sect. III and in [7].

ACKNOWLEDGMENT

Thanks to Manfred Broy, Humberto Nicolas Castejón, Øystein Haugen, Frank Alexander Kraemer, Mass Soldal Lund, Birger Møller-Pedersen, Ragnhild Kobro Runde, Richard Sanders, Ina Schieferdecker, Ketil Stølen, and Thomas Weigert for commenting on earlier versions of this paper.

REFERENCES

[1] J. E. Y. Rossebø and R. Bræk, "Towards a framework of authentication and authorization patterns for ensuring availability in service composition," in *The First International Conference on Availability, Reliability and Security (ARES 2006)*, April 2006.

[2] *UML 2.0 Superstructure Specification, formal/05-07-04*, Object Management Group, 2006.

[3] R. Sanders, H. N. Castejón, F. Kraemer, and R. Bræk, "Using UML 2.0 collaborations for compositional service specification," in *Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, October 2005.

[4] R. T. Sanders, R. Bræk, G. von Bochmann, and D. Amyot, "Service discovery and component reuse with semantic interfaces," in *Proceedings of the 12th International SDL Forum (SDL 2005)*, June 2005.

[5] J. E. Y. Rossebø, M. S. Lund, K. E. Husa, and A. Refsdal, "A conceptual model for service availability," *Quality of Protection: Security Measurements and Metrics*, vol. 23, August 2006.
[6] W. A. Arbaugh, W. L. Fithen, and J. McHugh, "Windows of vulnerability: A case study analysis," *IEEE Computer*, vol. 33, no. 12, pp. 52– 59, 2000.
[7] J. E. Y. Rossebø and R. Bræk, "Towards a framework of authentication and authorization patterns for ensuring availability in service composition," Research report 332, Department of Informatics, University of Oslo, April 2006.
[8] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP authentication: Basic and digest access authentication," RFC 2617, June 1999.
[9] B. Schneier and D. W. Mudge, "Cryptanalysis of Microsoft's point-to-point tunnelling Protocol (PPTP)," in *Proceedings of the 5th ACM Conference on Communications and Computer Security*, November 1998.
[10] *ISO/IEC 15408, Information technology – Security techniques – Evaluation criteria for IT Security*, International Standards Organization, 1999.
[11] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli, *Role-Based Access Control*. Artech House, 2003.
[12] W. Tolone, G. J. Ahn, T. Pai, and S. P. Hong, "Access control in collaborative systems," *ACM Computing Surveys (CSUR)*, vol. 37, no. 1, March 2005.
[13] I. Ray, L. Na, R. France, and K. Dae-Kyoo, "Using UML to visualize role-based access control constraints," in *Proceedings of the Ninth ACM symposium on Access control models and technologies (SACMAT 2004)*. ACM Press, June 2004.
[14] E. C. Lupu and M. S. Sloman, "Conflicts in policy-based distributed systems management," *IEEE Transactions on Software Engineering*, vol. 25, no. 6, November-December 1999.
[15] H. N. Castejón and R. Bræk, "Dynamic role binding in a service oriented architecture," in *Proceedings of the 2005 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM)*, vol. 190. Springer-Verlag, October 2005.
[16] Z. Fu, F. Wu, H. Huang, K. Loh, F. Gong, I. Baldine, and C. Xu, "IPsec/VPN security policy: Correctness, conflict detection, and resolution," in *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, January 2001.
[17] J. D. Moffett and M. S. Sloman, "Policy conflict analysis in distributed systems management," *Journal of Organizational Computing*, vol. 4, no. 1, January 1994.
[18] E. Lupu and M. Sloman, "Reconciling role based managment and role based access control," in *Proceedings of the 2nd Role Based Access Control Workshop*, November 1997.
[19] S. Reiff-Margeniec and K. J. Turner, "A policy architecture for enhancing and controlling features," in *Proceedings of Feature Interactions in Telecommunication Networks VII*, June 2003.
[20] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "Ponder: A language for specifying security and management policies for distributed systems," Imperial College Research Report DoC 2000/1, Department of Computing, Imperial College of Science, Technology and Medicine, October 2000.
[21] J. Floch and R. Bræk, "A compositional approach to service validation," in *Proceedings of the 12th International SDL Forum (SDL 2005)*, June 2005.
[22] R. T. Sanders and Bræk, "Modeling peer-to-peer service goals in UML," in *Proceedings of the 12nd IEEE International Conference on Software Engineering and Formal Methods (SEFM 2004)*, September 2004.
[23] F. B. Engelhardtsen and A. Prinz, "Application of stuck-free conformance to service-role composition," Presented at the 5th Workshop on system analysis and modelling (SAM 2006), June 2006.
[24] O. Halvorsen and O. Haugen, "Proposed notation for exception handling in UML 2.0 sequence diagrams," in *Proceedings of the 17th Australian Software Engineering Conference (ASWEC 2006)*, April 2006.
[25] *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, formal/06-05-02*, Object Management Group, 2006.
[26] R. Bræk, K. E. Husa, and G. Melby, "ServiceFrame: WhitePaper," August 30, 2006 [online] – URL : http://ikt.hia.no/teleservice/ServiceFrameWhitepaperv8.pdf, April 2002.
[27] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security," Presented at the 4th Conference on Pattern Language Programs (PLoP97), August 1997.
[28] B. H. B. Cheng, S. Konrad, L. A. Cambell, and R. Wassermann, "Using security patterns to model and analyze security requirements," in *Proceedings of the 2nd International Workshop on Requirements for High Assurance Systems*, September 2003.
[29] F. L. Brown, J. Divietri Jr., G. D. Villegas, and E. D. Fernandez, "The authenticator pattern," Proceedings of Pattern 6th Language Programs (PLoP99), August 1999.
[30] E. D. Fernandez and R. Warrier, "Remote authenticator/authorisor," Presented at the 10th Conference on Pattern Language Programs (PLoP2003), August 2003.

Judith E. Y. Rossebø is a Research Scientist at Telenor Research and Innovation. Prior to joining Telenor in 2000 she worked three years as a systems engineer at Alcatel Telecom Norway and one year as an assistant professor teaching mathematics at the University of Tromsø. At Alcatel she worked with dimensioning, performance, dependability and traffic control in telecommunication networks. She received a cand.scient. degree in Mathematics from the University of Oslo in 1994 and is currently working on a PhD at Norwegian University of Science and Technology (NTNU), department of Telematics, in the SARDAS project. Since January 2003 she has been the Chairman of ETSI TISPAN WG7 Security. Her research interests include security in general; security issues in multimedia communications services, and in particular securing availability of services.

Rolv Bræk received his Siv.ing. degree (M.Sc.) in 1969 from the Norwegian Institute of Technology (NTH), Trondheim, Norway, and is currently Professor at the Norwegian University of Science and Technology (NTNU), the Department of Telematics. He has previously been director of research at Sintef. Bræk has extensive experience from development of communication control systems using formal methods. He has developed methodology based on SDL and participated in the SDL standardization work within ITU-T as well as in tools development for SDL. He is one of the initiators of the PATS collaboration between Telenor, Ericsson, Compaq and NTNU on service arcitectures, service execution frameworks and service platforms for hybrid services. His current main research interest is rapid, model driven service engineering.