TOWARDS DYNAMIC COMPOSITION OF HYBRID COMMUNICATION SERVICES

Jacqueline Floch and Rolv Bræk
Department of Telematics, NTNU
N-7491 Trondheim, Norway
e-mail: {Jacqueline.Floch, Rolv.Braek}@item.ntnu.no

Key words: Architecture, Telecommunication, Services, Service models, Service composition, Service execution framework, Roles, Role assignment, Role learning.

Abstract: Due to the deregulation of the telecom network and the Internet, users will have access to an increasing number of heterogeneous communication services and will need to adapt their services or learn new services in order to interact with other users and systems. We propose a dynamic composition method that enables services to be constructed dynamically or "on-the-fly" from existing functional elements (service roles). Roles and actors that play roles are key concepts in our approach. A service role is defined as the part an object takes in a service. Service execution requires that roles are assigned to actors in a coordinated way. Our approach enables the systematic and structured specification of services, and provides mechanisms for service composition and an execution environment.

1 INTRODUCTION

Since short time to market for new services is increasingly important, the telecom operators and manufacturers are in constant search for better frameworks and methods for the rapid construction and deployment of services. Frameworks such as the TINA Service Architecture [1] or the programmable architecture from EPFL [2] have been proposed. We believe that a traditional approach to service life-cycle as proposed by TINA with the associated scenarios for service deployment, user subscription and service withdrawal, is not flexible enough to future market expectations and service providers needs. Users expect to access a similar set of services independently of what network they happen to use, they expect to get access to new and useful serv-

ices as they become available, and they expect to be able to communicate seamlessly with other users and applications regardless of what service provider and network operator they use. Conversely, service providers need to develop and deploy services across the network technologies at a pace and with a degree of harmonisation that keeps up with user expectation. As the open world of Internet enables several parties to develop and provide rapidly new services, as new actors are emerging in the deregulated telecom network, users will have access to an increasing number of heterogeneous services and will need to adapt their services or learn new services in order to interact with other users and systems. The mobility of users also creates a need for adaptation. For example, UMTS (Universal Mobile Telecommunication System) introduces the Virtual Home Environment (VHE) and service portability. This allows users to access their personalised services in any visited network in the same way as in their home network, even if the services are not actually offered in the visited network.

Hybrid communication services are sometimes defined as services that span different network technologies including the public switched network (PSTN) and the Internet [3]. We also consider as hybrid services, services that are provided by interactions between *heterogeneous* service components, developed by different parties or made available by different service providers. In our work, the latter kind is in focus. Our goal is to define a framework for service execution and methods for service design that enable services to be designed separately and then composed dynamically using Plug-and-Play techniques [4].

Service design is complex. Communication services normally require the coordinated effort of several distributed components, where some of the components may be involved in several services. Services involve several users where some of them may participate in several services. In addition to this intricate structure of services, the need for service inter-operability across several operator domains and for service availability over different network technologies also contribute to the complexity of service design. Our approach to service design is based on *service roles* [5]. We define a service role as the part an entity (an object in our framework) takes in a service, and we consider a service as a collaboration between roles. By using services roles, we are able to break down the complexity of service specification, and also to combine roles and provide new services in a flexible way.

Dynamic service composition enables hybrid services to be constructed dynamically from heterogeneous service roles. The method supports service adaptation and learning, and thus provides service users and providers with the ability to build new services “on-the-fly”.

This paper presents our approach to service development and execution based on role modelling and dynamic service composition. The next section introduces the basic concepts: *roles* and *actors*. In Section 3 we present the

service execution framework in which we are experimenting with the dynamic composition method. The purpose of the framework is to enable the systematic and structured specification of services by defining concepts for service modelling. The framework also provides basic mechanisms for service composition and an execution environment. In Section 4 we present some examples that illustrate how dynamic composition may be used during service execution. The examples also introduce the concepts of *role assignment*, *negotiation* and *learning*. Section 5 presents the main ideas in our approach and discuss potential solutions. We identify the requirements that the approach sets on the service development. In Section 6 we discuss composition correctness. Finally we compare our approach to some related work.

2 ACTORS AND ROLES

Actors and service roles are key concepts in our framework. A service role is the part an entity (an object in our framework) takes part in a service. Thus a service is seen as a collaboration between roles [5]. Service execution requires that roles are assigned dynamically to objects (that we denote as actors) in a coordinated way.

Roles may be introduced in a rather intuitive way. The concept of role is used extensively every day, either to describe relations between persons, for example family roles such as mother and daughter, or to describe functions and responsibilities, for example organisational roles such as professor, secretary, librarian, student. Also in telephony, it is customary to refer to the A-subscriber and B-subscriber, or the caller and the callee in a call.

The concept of role was already introduced in the end of the 70's in the context of data modelling [6] and has emerged again in the object-oriented literature. Roles are used both for data modelling [7] and functional modelling [8], [9], [10]. In our approach, service roles are functional roles that encapsulate the functional properties of components involved in a service.

As services involve several distributed components where some of them may be involved in several services, using roles enables to focus on single "slices" of behaviour. This is illustrated in Figure 1. This figure shows a service role collaboration diagram for basic call. It also shows an underlying object structure implementing the roles. In this figure two User Servers are involved that play the subscriber roles "caller basic call" and "callee basic call". The User Servers may play different roles in other services as shown in Figure 2. Here one of the User Server plays the role "call forward". This service role collaboration may occur if the user B initially called by A is busy or does not reply.

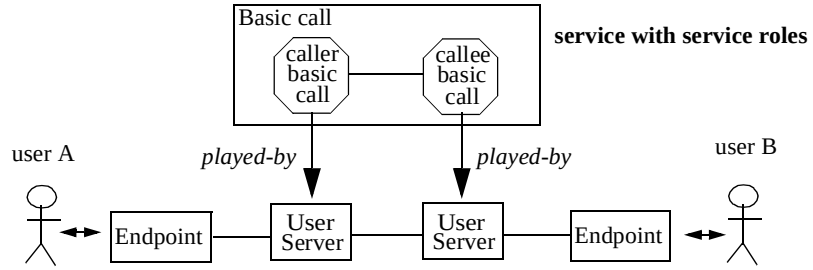

*Figure 1.*Service role collaboration for basic call.

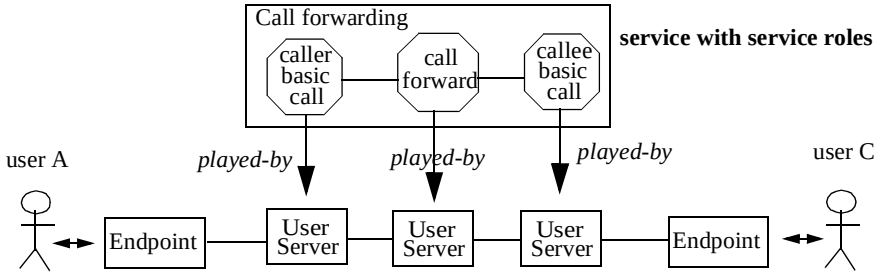

*Figure 2.*Service role collaboration for call forward.

These examples illustrate that roles may be combined in different manner. The role “caller basic call” may both collaborate with “callee basic call” or “call forward”. The roles must be specified such that they collaborate in a consistent manner. In addition the collaboration of roles should not conflict or interfere in an undesirable manner. In the call forward example, we may check that the user A does not object to be forwarded to C.

While these two first examples show quite simple service role collaborations, some service demand more complex collaboration structures. This is shown in Figure 3. Here one of the User Server plays concurrently the two roles “callee basic call” and “call waiting” towards the other User Servers. Some coordination between these roles may be required.

Although these examples use telecommunication services, role modelling can also be applied to information or management services. Telecommunication services are especially interesting because they usually involve several interacting roles played by concurrent objects.

Role modelling is not new. The originality of our work is that we use roles for communication services and that we assign roles and compose them dynamically. In our approach, actors are objects that are created and assigned service roles dynamically on demand. A user may participate in several independent services or service sessions. In that case, several concurrent actors are created and allocated the independent services roles. Actors may change

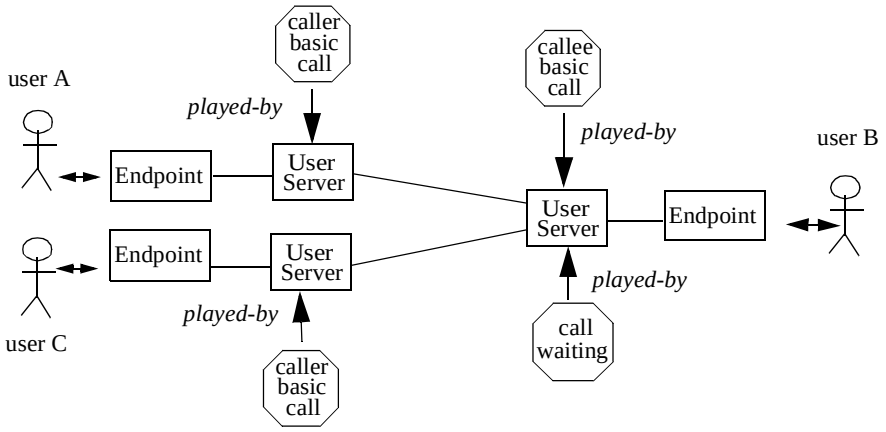

*Figure 3.*Service role collaboration for call waiting.

roles during their lifetime. Within each service session, an actor may also play several roles. These roles are usually functionally related or they may share some common resources (e.g. a terminal).

The assignment and composition of roles must be coordinated in order to ensure a correct behaviour. For this purpose, actors are created with some intrinsic behaviour required for role coordination. We distinguish between the intrinsic behavioural properties of an actor i.e. the properties of the actor itself, and the extrinsic properties that are obtained through the role assignment [9].

In resume, the main issues in the work presented here are the definition of rules for the specification of composable service roles, the assignment of roles to actors and their coordination within one actor.

3 EXECUTION FRAMEWORK

We have introduced a simple execution framework in which we are experimenting with dynamic composition. The purpose of this framework is twofold: 1. to enable the systematic and independent specification of services by defining concepts for service modelling; 2. to provide basic mechanisms for service composition and execution.

Our framework is inspired by the Telecommunication Information Network Architecture (TINA) [11]:

- We use a wide definition for service that includes telecommunication, management and information services.
- We separate between service logic and network resources and protocols.
- The components of the architecture are defined and implemented as interacting objects, following a distributed object oriented approach.

- We use the concept of session where a session is a means for representing a relationship between a group of objects collaborating to a service.
- We assume that the components execute in a distributed processing environment that support locating objects and interaction between objects.

In order to focus on the composition problem, we have introduced several simplifications to TINA. While TINA proposes a network-centric approach where the central service components belong to the provider domain, we do not make any assumption about the domain where our components are being deployed. We also simplify the TINA session model and let the user session components play the role of the session manager. This latter simplification restricts mobility and multi-party support, but it also considerably simplify the service usage scenarios and facilitate our experimentation.

Figure 4 shows a simplified UML collaboration diagram where the main computational service components (or objects) in our framework are introduced. The components are defined in a generic manner and need to be specialized for specific services. The generic functionalities of the components are defined as follows:

- The **User Endpoint** represents different interfaces that provide the user with service access and usage such as a telephone set, a personal computer or any other terminal equipment. The user may be a human user or an application.
- The **User Agent** is a service independent component that represents the user and the user profile in the network. It records information about the user subscription to services, the endpoints where the user can be reached, the user preferences and the user participation into services. The User Agent is involved during the instantiation of new service sessions and is in charge of role assignment. The User Agent becomes increasingly important when services become more personalised.
- The **Session Actor** is a service independent component that can play different service roles. The Session Actor coordinates the execution of concurrent roles. A service role is assigned to a Session Actor either when a user requests to participate in a service session (e.g. “caller” role in a basic call), or when a user is invited to participate in a service session (e.g. “callee” role in a basic call or “call waiting” when busy), or when some special events happen during a service session (i.e. “billing” after session control negotiation, or “announcement” before the initiation of call forward).
- The **Communication Endpoint** is a service independent component. It coordinates the establishment of stream flow connections. This component is not detailed in our framework. It represents the interfaces to the transport network resources.

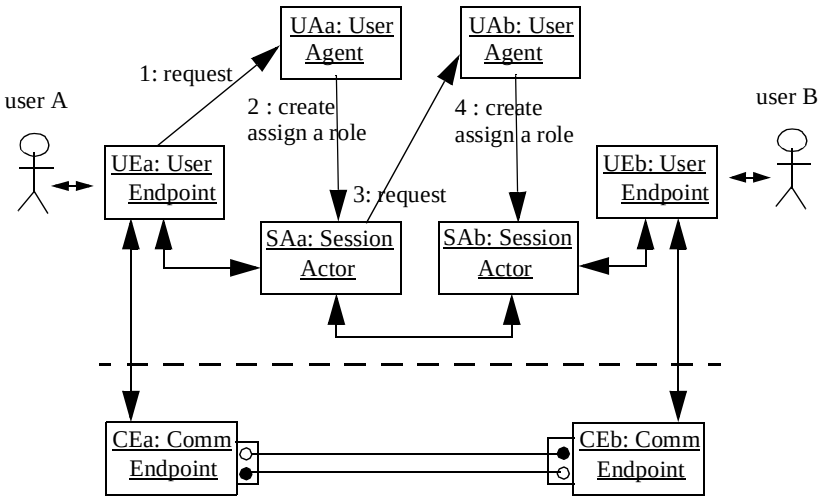

*Figure 4.*Service session model.

The computational objects execute in a distributed processing environment (DPE) such as CORBA or Java RMI that mainly support communication and binding with remote objects. Through the definition of the User Agent and Session Actor, and the allocation of responsibilities to these components, we confine service specification enabling the service designer to focus on service logic, role interaction and consistency. In this way our framework provides additional functionality to a DPE.

4 EXAMPLES

Our aim with the following examples is to:

- demonstrate the potential of dynamic composition,
- describe different forms of composition (e.g. sequential or parallel composition),
- identify the basic elements or steps in the composition process and,
- capture the basic requirements to the composition method.

We present three examples. The first example introduces the concept of role *assignment*, the second the concept of *adaptation* and the third the concept of *learning*. The examples are presented using UML sequence diagrams that describe the interactions between the system components [12]. Several alternative interactions are often possible; we present one of them. The aim is

not to propose an optimal service design, but rather to identify particular needs in the composition approach. Again here as in Section 2 and for the same reasons, we use simple telecommunication services.

4.1 Role assignment in a Basic Call

This example is given as introduction to the concept of role assignment to Session Actors. It shows that we consider a service as a collaboration between roles and that these roles are assigned to actors at service execution. The example involves two users A and B where user A is making a basic call to B. See Figure 5.

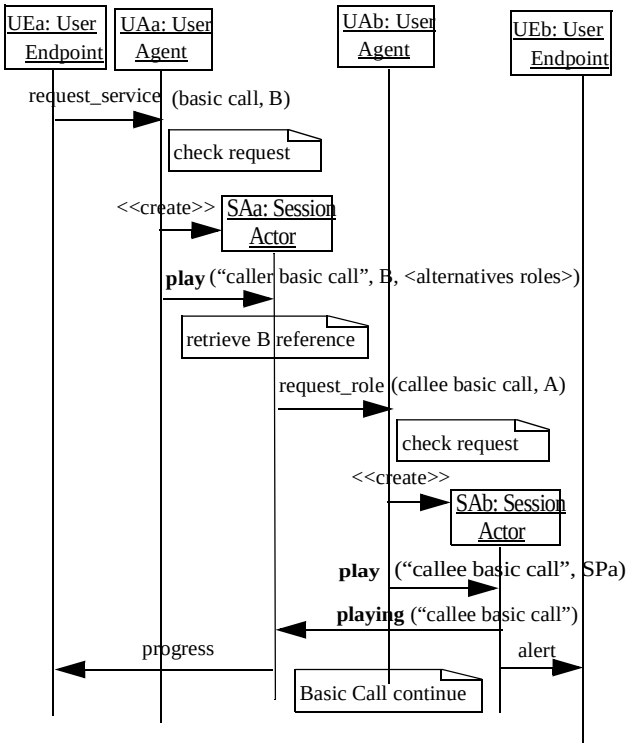

*Figure 5.*Role assignment in Basic Call.

In this example a Session Actor is created and assigned the "caller basic call" role by the User Agent on A side when the user A requests to participate in the service. The retrieval and loading of role behaviours are basic functionalities supported by the distributed processing environment. The new Session Actor locates the User Agent that represents B and invites the user B to participate in the service. A new Session Actor is created and assigned the requested "callee basic call" role by the User Agent on B side. Then the service can continue.

The assignment of roles must be coordinated in order to avoid inconsistencies or undesirable service interaction. The User Agent serves to coordinate the assignment. It also checks the request against the user's preferences and subscription profile.

When assigning the role to Session Actor on A side, the User Agent may specify a list of alternative roles. This information indicates to the Session Actor which other roles may be played if the current request cannot be honoured. Role compatibility information, conditions and priorities may be associated to alternative roles. For example an alternative role may be associated to the condition "B busy". Using the specification of alternative roles, the Session Actor may decide on behaviour changes without involving the User Agent. This will be shown in the next example.

The messages "play" and "playing" are service role independent. We will introduce other role assignment related messages and interactions in the next examples. We define these interactions such that role assignment can be clearly distinguished from playing the role itself.

4.2 Call Forward when Busy

This example illustrates the ability to adapt a service session to a specific situation by using an alternative behaviour. It involves three users A, B and C. The user A wishes to establish a basic call to a busy user B. Incoming calls to B are forwarded to C if B is busy. The example shows that several Session Actors may co-exist and execute in parallel on one user side (here on B side).

On request from A, the User Agent for B determines that the "call forward" role should be played (see Figure 6). This role is played concurrently from the role currently played at B and is assigned to a new Session Actor. The Session Actor on the A side is informed about B playing this new role; this gives the Session Actor the opportunity to accept or reject the role, but also to check for call barring cases, etc. before service execution continues.

We assume that the Session Actor on A side was given alternative roles during its initial role assignment (see Figure 5). Thus it knows that the current role "caller basic call" is compatible with the new role "call forward" and that the assignment of a new role on A side is not necessary. We will see in the next example that A may also reject the role proposed by B and negotiate other roles with B.

4.3 Learning alternative behaviour

This example illustrates the ability to enhance a service by learning a new behaviour. It involves two users A and B. The user A wishes to establish a basic call to a busy user B. A and B negotiate a behaviour for handling the

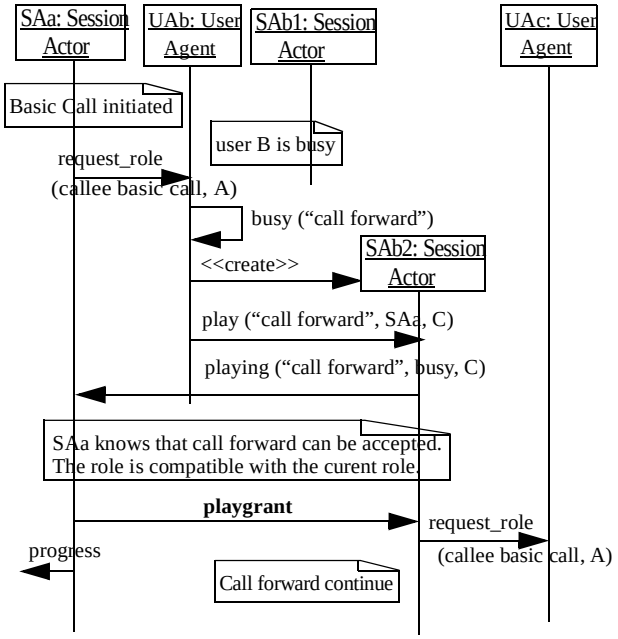

*Figure 6.*Service adaptation and parallel role assignment.

busy situation. We assume that different services are available on A and B sides. This would be the case when A and B are subscribers in different networks.

On request from A, the User Agent for B determines that the "call waiting" role should be played (see Figure 7). The "call waiting" role permits to notify the user B about the incoming (waiting) call and to let user B choose to accept or reject the waiting call. This role has to be tightly synchronized with the role currently played by B, so it is allocated to the existing Session Actor. The Session Actor on the A side does not agree on this role (the user A may wish to not interrupt B) and proposes to initiate the alternative "call back" role. This service role alternative is not available on B side where the Session Actor has to be learned the new service role. Learning includes the downloading of code to the Actor.

We assume that both Session Actors were informed about alternative roles and assignment conditions during their initial role assignment and are able to decide about behaviour changes without involving the User Agents. The Session Actors inform the User Agents about behavioural changes (sending the message "playstatus"). In some cases the Session Actors may need external support to make decision about the assignment of a new role. Decision support may be provided by the User Agent, some other specialized agent in the network or the end user.

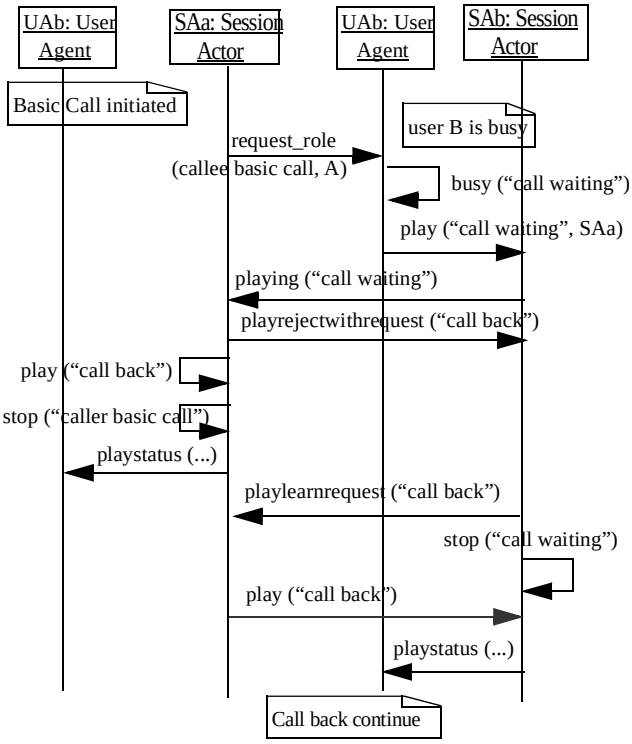

*Figure 7.*Learning a new role.

The “call back” service role has to be coordinated with the role currently played by the Session Actor on B side. The activation of “call back” is delayed until some synchronization point is reached in the role played by B, for example the suspension of the role or its completion. Some conditions may be attached to the synchronization points. A synchronization condition could specify that any voice based service should be delayed after completion.

4.4 Other examples

The composition approach may also be used in other situations and applied to information and management services:

- Filters or translators may be used in order to adapt a service to a user or an application. For example, a user may prefer not to be notified of the arrival of electronic messages in the morning, except from those sent from some selected sources.
- Services may also be adapted to handle some specific type of information. For example, the information retrieved from a name directory may have to

be re-formatted. Translation from voice to text or from text to voice may also be needed.
- A service may be adapted to some particular equipment. A subscriber of a PSTN that is using a traditional telephone set with no processing capability may need to interact with an electronic service support application that was designed for interaction with remote PC users. Some adaptation function based on an IVR may be introduced between the telephone user and this electronic service.
- Billing, logging and other management functions may be designed as service roles that are composed with telecommunication and information services.

5 ROLE BASED SERVICE DEVELOPMENT

5.1 Specifying roles as state machines

Service roles make it possible to break down a complex behaviours into simpler ones. A service role is the part that an object plays in a service. Each object plays (communicates) with other objects that play dual or consistent service roles. By dual roles we mean that roles complement each other and can play together in a correct way, e.g. “caller” and “callee” roles in a call. By consistent roles, we mean that roles can be aligned in order to provide a consistent behaviour (see Section 6).

We use the same mechanisms for designing role types as for classes. Interactions between roles can be modelled using message sequence charts and their detailed behaviour using state machines. For example the behaviour of roles may be specified using MSC [13] and SDL [14]. A main difference between role types and classes is that role types cannot be instantiated on their own but need to be assigned to objects. Role assignment sets requirements on the structure and behaviour of the object being assigned a role. This will be discussed later in this paper.

5.2 Forms of composition

Service roles are composed in order to provide advanced service functionality. We distinguish between different forms of composition:
- Sequential composition: service roles are executed in a sequential order. Information may be transferred between two consecutive roles. For example, the “basic call” and the “call back” service roles may be composed serially.

- Parallel composition: service roles are executed in parallel. The roles may be executed for the same period of time or they may start and end at different times. The “billing” and “call basic” service roles may be composed in parallel with billing being started when “call basic” reaches a connected state.
- Alternating composition: service roles are executed in an alternate manner i.e. only one role is active while the others are passive waiting for some event that enables them to be activated. Synchronization points are points of the role execution where the activation, suspension and resumption of the role execution can take place. Conditions may be associated with the synchronization points. In that case, the activation (or suspension and resumption) of the role execution can only take place if the conditions are fulfilled. Actions to be performed at role activation (or suspension and resumption) may also be attached; for example, communication resources may be released at suspension and reallocated at resumption. The “basic call” and “call waiting” service roles may be composed in an alternative way.

5.3 Composition behaviour

Our approach requires that roles are designed in order to enable composition. Roles must be prepared to report and catch events related to composition, and to coordinate with other roles they are composed with. We define separately the composition behaviour and the service behaviour. Roles are assigned, negotiated, learnt, suspended and resumed using interactions specific to the composition behaviour and independent from the service role behaviour.

Role composition is triggered and controlled by service dependent events, by user states (e.g. idle, connected, busy) and user preferences. In order to generalize composition triggering and to define patterns for role design, we need to identify and classify triggering events. We reuse the knowledge from other frameworks where important service events have been identified, e.g. Parlay [15].

Role assignment, negotiation and suspension may be attached service dependent information; this is done using service independent mechanisms. As shown in the previous examples, a list of alternative roles (and associated conditions and actions) may be specified for role negotiation. The contents of the list depend on the service role, but the mechanisms defined for specifying and interpreting the list are service independent. For example, in “call forward when busy”, we may specify that a check for call barring cases (this is a service dependent action) has to be performed before accepting the “call forward”.

5.4 Synchronization

Alternating composition requires roles to be tightly coordinated with other roles. Coordination occurs at synchronisation points. As dynamic composition may be applied to service roles that are designed by different parties and at different times, it may not be possible, when designing a role, to foresee the behaviour of the roles this role will be composed with. In order to generalize the coordination of alternate roles, we need to identify potential synchronisation events. Here again, we reuse the knowledge from other frameworks where important service events have been identified, e.g. Parlay [15].

5.5 Role manager

The role manager encapsulates the intrinsic behaviour of a Session Actor. It supports the assignment of service roles to actors, and coordinates the negotiation and the learning of new roles. The role manager checks and initiates the conditions and actions attached to service assignment and negotiation. In addition, it coordinates the synchronisation of alternate roles.

While the User Agent coordinates a set of actors acting for a user, the role manager coordinates a set of roles within an actor. Thus our framework introduces two levels of control, one at the actor level and one at the role level.

5.6 Negotiation decision

In the examples presented in Section 4, we have assumed that actors were able to decide whether a role should be granted or rejected, and to negotiate alternative roles. This decision was based on the interpretation of a list of alternative roles provided by the User Agent. The list is generated from a role map that describes relations between roles and correct execution orders (see Section 6). The role map can be adapted to user's preferences. For example, a user may prefer not to use "call waiting" when busy although this is a correct combination of service role and state.

If the information contained in the list of alternative roles does not permit the Session Actor to make a decision during negotiation, the Session Actor may require external support. Decision support may be provided by the User Agent or some other specialized agent in the network. The users may also be involved in the decision process. As some terminals with limited capabilities make it difficult to involve the users, the capabilities of the user terminals have to be taken into account. The preferences of the users involved in the service must also be significant. In our third example, the user A rejects "call

waiting" because A does not wish to interrupt B; when A proposes the "call back", it would be incoherent to interrupt B in order to let him decide if the role can be agreed.

5.7 Several actors and actor types

The examples presented in this paper have focused on the collaboration of roles assigned to two different Session Actors. We plan to develop cases where more than two Session Actors are involved.

In our work, we have chosen to first concentrate on the assignment of roles to Session Actors. We will further develop the method in order to support the assignment of roles to the other components of the framework e.g. the User Endpoint.

6 COMPOSITION CORRECTNESS

Although our work concentrates at the moment on the service design aspects, we also have in mind the need for ensuring service correctness. Service correctness is not a particular requirement set by the composition method. We meet similar problems in traditional service development and deployment approaches where the completeness and consistency of service specifications, and the conformance of implementations to specifications have to be ensured. However the assignment of roles to actors and their composition in our method require special attention.

Development based on formal methods enable the verification of the specifications, the automatic generation of code and test cases. It seems that few attempts to use formal methods on large scale have been done and that the methods are usually applied in general, not to the special case of hybrid communication services [16]. By breaking down complex service behaviours into small behavioural roles where each role can be specified using some formal method (e.g. SDL [14]), and by constraining the ways these small behaviours can be composed together, we intend to produce specifications and executable services that can be more easily verified and validated.

The role of the User Agent is crucial with respect to behavioural consistency. The User Agent coordinates the assignment of roles in order to avoid inconsistencies or undesirable service interactions. It also checks that the assignment is consistent with respect to the user's preferences and subscription profile. We can distinguish between two problem areas:

- the role alignment i.e. ensuring that the roles assigned to interacting Session Actors or negotiated between Session Actors collaborate in a consistent manner leading to the execution of a correct service. For example the

"caller basic call" role should interact consistently with the "callee basic call" role.
- the role execution ordering i.e. checking that roles are assigned and played in a correct order. For example, the call forward role can only be assigned in relation with some other role assignment request such as call diversion.

6.1 Role alignment

Each Session Actor plays (or provides) a role that requires the interacting Session Actor to play a consistent role. An association between actors is valid if the provided roles "contain" the required roles [5]. Roles being assigned should be aligned in order to ensure containment. We distinguish between three levels of assignment:
- Validation. Checking that the required roles are contained in the provided roles. Validation is illustrated in our first example where the "caller basic call" role and "callee basic call" role must be validated.
- Adaptation. The roles are negotiated and adapted. Adaptation is illustrated in our second example where the "call forward" role is adapted to the "caller basic call" role.
- Learning. A role can be learned when the existing roles on one side are insufficient. Learning is illustrated in our third example where the "call back" role is learnt by the B side.

Ideally, role alignment should be performed dynamically. This would enable new services to be produced in a flexible manner during service adaptation and service learning. In practice, the alignment is done during system design. When a service is designed, new service roles are defined that complement each other (dual roles) or new roles are designed to play with existing roles.

6.2 Role execution ordering

Service roles cannot be composed in any order. When a new role is introduced, it is necessary to relate this new role to other existing roles and describe execution constraints. We call this description a role map. The map specifies the allowed role sequences. A notation was introduced in [9] for specifying allowed role sequences. This notation needs to be further developed in order to enable annotations that describe events, conditions and actions to be associated to the sequences in the map.

7 OTHER ISSUES

We are aware that several other issues would need consideration. We have chosen to concentrate on the functional service aspects. We list some other interesting issues:

- Role discovery and downloading. Existing frameworks such as Java and Jini support retrieval and downloading. The Plug-and-Play project proposes a basic infrastructure for dynamic plug-in and develops a demonstration platform based on Java that enables dynamic component plug-in [4]. Our work is based on the concepts introduced by Plug-and-Play.
- Service subscription. The possibility to learn new service roles at execution time changes the way of thinking about service subscription (i.e. the right to access a service). Operators and services providers may wish to limit the access to new services.
- Service billing. In traditional telecommunication networks, billing is related to service subscription and use. Dynamic composition makes traditional billing difficult. Rules for billing services that are not part of a subscriber repertoire but are learned during service execution may be designed.
- Failure management. The services roles downloaded from other parts may lead to errors. Failure management support is needed that permits the control of failures for downloaded code.
- Upgrade. There may exist several versions of a service role. Support for deploying new versions, for identifying specific versions is needed.
- Security. The operators and service providers will require the control of new service roles with respect to security.
- Performance. The dynamic composition should not introduce waiting delays that are longer than tolerated by the service users.

8 RELATED WORK

As explained in Section 3, our framework is inspired by TINA [11]. We extend this approach in order to provide a flexible composition of services. We have introduced simplifications to the TINA session model, that facilitate our experimentation.

The notion of role is currently used in several object oriented methodologies. [9] describes the properties of roles. It introduces the concept of “roleification” and compares it to other abstraction concepts as specialization and aggregation. It discusses the specification of behavioural roles and present how roles can be attached to objects. It also defines a notation for the representation of allowed role sequences. We propose to further develop this nota-

tion in order to specify role maps. OORASS is an object oriented methodology based on role analysis and synthesis [8]. Roles are used in the analysis that allows the designer to break the total problem into sub-problems. Synthesis combines the role in order to produce objects. While OORAS has been mainly applied to information systems, we focus on communication services. In addition, we propose a dynamic synthesis of roles.

A service composition method is proposed in the CANES project [17]. The method addresses transport services in Active Networks i.e. Internet based networks where packets may contain executable programs that are delivered to network elements. The services discussed in CANEs are related to transport, e.g. packet forwarding, packet filtering, transcoding. Our work focuses on high level services that have a more complex structure than transport services. The “hybrid” aspect is not really important in CANEs and is not discussed.

Several execution frameworks are available (e.g. CORBA, Jini, DCOM) that propose general solutions to component based systems. In these frameworks, the components are described by static interfaces. Thus the interfaces do not cover the dynamic behaviour and cannot be checked in order to guarantee the correctness of interactions between components that are dynamically bound together. [18] introduces to several architecture concepts and investigate the properties that interfaces should provide in order to guarantee the correctness of connections.

Feature interaction is a fundamental problem in service creation. The problem arises from the extension of telecommunications system functionality, feature by feature. In [19] the authors try to define a modular approach to service specifications. Each feature is implemented by one or two components types, and each external call is processed by a dynamically assembled configuration of components. The proposed configuration is limited to an assembly of pipes and filters, and the interactions modules are strictly limited by the architecture. Our architecture is less restrictive than the pipe-and-filter based architecture. We believe that the coordinated composition of service roles and the definition of role maps are means for avoiding undesirable interactions.

9 CONCLUSION

We have proposed an approach and a service execution framework that enable services to be constructed dynamically from heterogeneous functional components or service roles. Using roles, we are able to break down the complexity of service specification. The service execution framework supports

service negotiation and learning. Compared to traditional service development and deployment methods, our approach offer several potential advantages:

- Roles encapsulate a limited set of functional properties. They are small and can be easily understood.
- The method enforces systematic design. Roles are defined using generic composition patterns.
- Service role implementations may be shared between users.
- Service behaviour may be negotiated and adapted to the special needs of users.
- New service roles can be easily be tested.
- Rapid and incremental deployment of new services is enabled.
- The allowed sequences of roles can be described. These sequences can be adapted to user's preferences. Undesirable service interactions may also be avoided.
- The approach is applicable to different types of services such as telecommunication, information and management services.

ACKNOWLEDGMENT

Our work is done in the frame of the Plug-and-Play project supported by the Norwegian Foundation for Research (NFR) [4].

We would especially like to thank you Humberto Castejon for inspiring discussions. Humberto, MSc. student at ETSIT, Universidad Politécnica de Valencia, has been working with service role modelling and implementation in our institute in the frame of his MSc. thesis.

REFERENCES

[1] TINA-C deliverable: Service Architecture Version 5.0. Available at http://www.tinac.com/ (accessed February 2000).

[2] C. Gbaguidi, J.P. Hubaux and M. Hamdi. "A programmable Architecture for the provision of Hybrid Services," IEEE Communication Magazine, July 1999.

[3] G. Vanececk et al. "Enabling Hybrid services in Emerging Data Networks," IEEE Communication Magazine, July 1999.

[4] F.A. Aagesen et al. "Towards a Plug and Play Architecture for Telecommunications," in Proc. of the Fifth International Conference on Intelligence in Networks, November 1999. Kluwer Academic Publishers. Information available at http://www.item.ntnu.no/~plugandplay/ (accessed February 2000).

[5] R. Bræk. “Using Roles with Types and Objects For Service Development,” in Proc. of the Fifth International Conference on Intelligence in Networks, November 1999. Kluwer Academic Publishers.

[6] C.W. Bachman and M. Daya. “The role concept in data models,” in Proc. of the Third International Conference on Very Large Databases, 1977.

[7] R. Wueringa and W. de Jonge. “The identification of objects and roles - Object identifiers revisited-,” Technical Report IR-267, Faculty of Mathematics and Computer Science, Vrije Universuteit, Amsterdam., December 1991.

[8] T. Reenskaug and al. “OORASS: Seamless support for the creation and maintenance of object oriented systems.” Journal of object-oriented programming, 1992.

[9] B.B. Kristensen and K. Østerbye. “Roles: Conceptual Abstraction Theory and Practical Language Issues,” Theory and Practice of Object Systems, Vol. 2(3), 1996.

[10] M. Mezini and K. Lieberherr. “Adaptative Plug-and-Play Components for Evolutionary Software Development,” in Proc. of the 13th Conference on Object-Oriented Programming Systems, Languages and Applications, October 1998.

[11] TINA. Information available at http://www.tinac.com/ (accessed February 2000).

[12] OMG Unified Modeling Language Specification. Version 1.3 June 1999. Available at http://www.omg.org/ (accessed December 1999).

[13] Recommendation Z.120, ITU Message Sequence Charts (MSC), October 1996. ITU-T.

[14] Recommendation Z.100, ITU Specification and Design Language (SDL), June 1994. ITU-T.

[15] Parlay API Specification 2.0: Generic Call Control Service Data Definitions. Available at http://www.parlay.org/ (accessed February 2000).

[16] X. Logean, F. Dietrich and J.P. Hubaux. “On Applying Formal Techniques to the Development of Hybrid Services: Challenges and Directions,” IEEE Communication Magazine, July 1999.

[17] E. Zegura (lead author). “Composable Services for Active Networks. Georgia”. Architecture document from the CANEs project, Georgia Institute of Technology. Information available at http://www.cc.gatech.edu/projects/canes/ (accessed Mars 2000).

[18] D. Luckham and al. “Three Concepts of Architecture”. Stanford University Technical Report CSL-TR-95-674, July 1995.

[19] M. Jackson and P. Zave. “Distributed Feature Composition: A Virtual Architecture for Telecommunications Services,” IEEE Transactions on Software Engineering, October 1998.