Unified Telecom and Web Services Composition: Problem Definition and Future Directions

Gregory Bond
AT&T Labs—Research
Florham Park, NJ, USA
bond@research.att.com

Ioannis Fikouras
Ericsson Research
Stockholm, Sweden
ioannis.fikouras@ericsson.com

Eric Cheung
AT&T Labs—Research
Florham Park, NJ, USA
cheung@research.att.com

Roman Levenshteyn
Ericsson Research
Aachen, Germany
roman.levenshteyn@ericsson.com

ABSTRACT
In the domain of web applications, the service-oriented architecture (SOA) promotes the composition of coarse-grained web services to build more complex web applications using standards such as WS-BPEL. This paradigm encourages software modularity and re-use, language and platform independence, distributedness, and integration across enterprise boundaries. At the same time, in the domain of telecommunications, standards such as the IP Multimedia Subsystem (IMS) architecture and the SIP Servlet API enable the composition of individual SIP applications into complete services, reaping similar benefits. There are also ongoing efforts to integrate the two domains. The ultimate goal is to enable service providers to easily re-use existing services to build new converged web and telephony services that offer the users a rich and seamless experience. However, there are a number of problems associated with composing modular telecom and web services. In this paper we clearly define these problems and explain how approaches to telecom service composition and web services composition are fundamentally different. By way of pointing towards solutions we critically examine a number of existing partial solutions to these problems and discuss how and why they fall short of providing a complete solution. Finally we discuss some future directions that research can take in order to completely solve these problems.

1. INTRODUCTION
Modular and reusable software components are key elements in the rapid development of customized services. In the domain of web applications, the service-oriented architecture (SOA) promotes the composition of coarse-grained web services to build more complex web applications using standards such as WS-BPEL. This paradigm encourages software modularity and re-use, language and platform independence, distributedness, and integration across enterprise boundaries. At the same time, in the domain of telecommunications, standards such as the IP Multimedia Subsystem (IMS) architecture and the SIP Servlet API enable the composition of individual SIP applications into complete services, reaping similar benefits.

Not surprisingly there exists a strong desire to compose web services and telecom features to form custom services that synergistically combine telecom functionality with the wealth of data accessible via web service APIs. In this paper, we call the composition of modular web services with modular telecom features “unified composition.” One example of unified composition could be a customized service that displays the social network personal profile of a caller (web service access) for an incoming call (telecom feature access).

In order to achieve the goal of unified composition, it is necessary to devise an approach that is at least as capable and flexible as each approach is on its own. However, devising such an approach has revealed itself to be a difficult problem given the complexities of the respective approaches to composition. In this paper we assert that the reason the unified composition problem has not been solved to date is because the problem has not been precisely defined.

In response, we undertake a detailed examination of the characteristics and requirements of the respective approaches to composition. By comparing the two approaches we are able to provide deep insights into the nature of the problem. Moreover, we explicitly state what problems need to be solved in order to achieve the goal of unified composition.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IPTCOMM ’09, July 7-8, 2009 Atlanta, Georgia, USA
Copyright 2009 ACM ...$5.00.

1.1 Terminology
Terminology used for the web services and telecom do-
mains differ in potentially confusing ways. For example in the telecom domain one thinks of a collection of "features" or "applications" composed by an "application router" comprising a composite "service." However, in the web services domain, one thinks of a collection of "web services" comprising an "application." Since this paper not only discusses each domain but also compares and contrasts them with the goal of unifying them, we have adopted a unified terminology that permits discussing comparable entities from either domain. In particular, we will use the term "module" to refer to a web service or a telecom feature, "composer" to refer to the entity that composes modules, and "composite service" to refer to the composed result. When we speak about the domains in a non-unified context we will use the traditional, domain-specific terminology.

2. BACKGROUND

2.1 Service-oriented Architecture and Service Composition

Service-oriented Architecture (SOA) is a set of design principles that allow efficient integration of heterogeneous systems and applications. A SOA abstracts the actual implementation, location and access mechanisms for the various software modules as explicitly addressable and programmatically accessible synchronous and asynchronous functions that expose a well-defined interface description. SOA services do not have business logic, protocol or state dependencies between each other and are therefore considered to be loosely coupled and stateless [19].

The abstraction from underlying mechanisms enables the decoupling of the application from the protocol and platform control logic and enables efficient reuse at the macro (service) level rather than micro levels (e.g. objects). This enables the developer to be more productive by focusing on developing the actual business logic of the application.

SOA applications utilizing Web Services (WS) standards are typically implemented using the find-bind-publish pattern (as shown in Figure 1). In this way of working an application that seeks to implement a location based weather forecast uses the Universal Description Discovery and Integration (UDDI) [24] protocol to discover available weather and location services based on their previously published interface description in Web Services Description Language (WSDL) [25] notation, then proceeds to "bind" i.e. execute (synchronous or asynchronous) the service via this interface in an RPC-like manner via the Simple Object Access Protocol (SOAP) [25] protocol. Any feature interaction issues are in this case resolved by the application designer at design time.

An architectural pattern used often to implement a SOA is the Enterprise service Bus (ESB). An ESB is a centralized message broker that abstracts different protocols and messaging systems and allows the transparent invocation of remote services regardless of their implementation. While an ESB clearly promotes loose coupling and is a useful part of a SOA, it does not represent a fundamental SOA concept and therefore will not be addressed any further in this paper.

2.1.1 Web Services Orchestration

Composite web services can be created through the aggregation of elementary services. In this context the terms orchestration and choreography have been introduced to describe the interaction of elementary services as part of a larger business process. A business process in this context is defined as a set of actions that have to be completed when the business receives an event. Consequently, managing such a process requires managing state over a longer period of time. Orchestration refers to a business process controlled by one of the parties as shown by the entity labeled "orch" in Figure 2 whereas choreography involves collaboration, in which each involved party describes the part they play [22].

The orchestrating entity controlling the execution of the business process implements the find-bind-publish pattern in order to use elementary services, while at the same time exposes a single abstracted service interface to the application that consumes it.

A typical example of a business process is the production process initiated by an inbound order including all its individual steps such as checking the credit status of the customer, accessing stock, manufacturing, etc.

For the purposes of comparing enterprise and telecom approaches to service composition we will focus on web services orchestration and its most prominent and mature example the Business Process Execution Language (BPEL) [5].

BPEL scripts are interpreted and executed by an orchestration engine that manages interaction between web services according to a predefined business process. This engine aggregates elementary web services and exposes them as a single web service. A BPEL engine is stateful; it maintains the state of all processes that it executes. The BPEL XML-based grammar enables the description of control logic for the participating web services.

A BPEL process consists of "primitive" and "structure"
activities. Some common primitive activities include: execute a web service (<invoke>), explicitly wait for a request (<receive>), create a new variable (<assign>), and defining a potential exception (<throw>). WSDL data types are used within a BPEL process to describe the information that passes between requests. Finally, WSDL can be used to reference external services required by the process. BPEL supports structuring a process as a sequence or a set of parallel flows and, in addition, also foresees conditional constructs as well as looping in such business processes.

2.1.2 Web Services Feature Interactions

Feature interactions problems were first investigated and are typically associated with the telecommunications domain. However, the feature interaction problem is not limited only to telecommunications. In principle, feature interactions may happen in any software system that is subject to change. Researchers investigating the problem of feature interactions in the web services domain claim that interaction is the very foundation of service-oriented architectures. Web services must interact and new composite services will ‘emerge’ from the interaction of more specialized services. As the number of available web services increases, their interactions will also become more complex. Many systems are built using third-party services, over which implementations the composer application has little control. Many web services interactions are intended, but others may be unintended and undesirable.

The research distinguishes between functional and non-functional interactions. Non-functional feature interactions usually affect service properties such as security, privacy or availability. Additional distinction is made based on causes of undesired feature interactions between web services. In particular, they have identified the following typical causes: goal conflict, resource contention, deployment and ownership, violation of assumptions, information hiding, policy conflicts and invocation order. Deployment and ownership and information hiding are named as web services-specific causes of interaction that are usually not encountered in closed, centralized telecommunications systems.

To understand how a negative feature interaction may occur in the web services domain, consider an example taken from [27]. An online bookstore is implemented by means of the web services technology. In particular, it uses a feature called Order Processing implemented as a reusable web service by its suppliers. When the bookstore invokes the Order Processing service of one of its Suppliers, this supplier will, in turn, place an order with one of its network of Other Suppliers if it does not have the requested book in stock. However, this can lead to a situation where the order is sent back to the bookstore itself, which is just an Other Supplier. If undetected, this can lead to an infinite loop of order requests, which could cause all actors linked via the loop to become unavailable. This is a feature interaction between two implementations of the same feature, Order Processing, which may lead to resource contention. This scenario is very similar to the familiar call forwarding loop scenario from the telecommunications domain.

Consider another example, highlighting the undesired feature interaction due to the invocation order problem. A buyer uses a composite web-application for online purchases. She is subscribed to two features (possibly implemented by 3rd party services) for her online purchases: encryption of purchase orders and payment information and logging of all orders and payments. If the encryption feature is used before the logging feature by the composite application, then the input of the logging feature becomes encrypted and therefore unusable. If logging is used before encryption, no such problem exists. This example also shows another feature interaction problem related to goals conflict and data hiding: while there is a trusting relationship between customer and online purchase platform, the relationships between customer and suppliers/partners of the purchase platform are untrusted and there is no guarantee that supplier of the purchase platform will adhere to the same privacy policy and not misuse the collected customer data for e.g. unsolicited ads.

2.2 Telecom Feature Composition

There are a number of approaches to telecom feature application composition (see [6] for a survey). However, only three approaches have been standardized for use with telecom over IP: at the architecture level the 3GPP IMS standard [2]; and within application servers the JAIN SLEE (Service Logic Execution Environment) [15], and the SIP Servlet API [16]. In this paper we consider only the SIP Servlet API, which standardizes a component called the Application Router (AR), a component responsible for application selection and composition [5]. The IMS architecture shares the same composition model as the SIP Servlet specification. However, the IMS S-CSCF operates at a coarser-grained level and the behavior of its SCIM (Service Capability Interaction Manager) [4] is not specified. As such, the SIP Servlet AR can provide the basis of a SCIM implementation. As for the JAIN SLEE specification, it has been passed over by industry in favor of the SIP Servlet specification.

2.2.1 Application Routing in SIP Servlet API

Application routing in the SIP Servlet standard is based on the theoretical Distributed Feature Composition (DFC) architecture [14]. When a SIP Servlet container receives an initial SIP request (i.e. a request that is not part of an existing SIP dialog or call), the container queries the application router (AR) to select an application to invoke. If the invoked application sends an initial request to continue the call, the container queries the AR again to select the next application to invoke. In this way, a chain of applications is formed until all the desired applications are invoked. Any subsequent requests and responses within established SIP dialogs are routed along the application chain, and the AR is not notified. Each invoked application operates as an independent SIP entity. It can send and receive SIP requests and responses, although in reality the message exchanges may be with the adjacent applications (upstream and downstream) in the chain executing in the same container. Figure 3 illustrates application composition in a SIP servlet container. Unlike a web services orchestrator, the responsibility of an AR is to compose a linear sequence of features for a given subscriber acting either as a caller or as a callee. As shown in the figure, a caller’s subscribed to features are composed prior to composing the callee’s features. An AR relies on four parameters in order to choose the next application for invocation.

1. The subscriber (caller or callee) URI from the initial request message.
2. The current routing region (e.g., originating or terminating). An application router must distinguish between regions because a subscriber’s features can differ between its originating (subscriber as caller) and terminating (subscriber as callee) regions.

3. The routing directive (NEW, CONTINUE, or REVERSE). A feature is free to specify the caller or callee URI in an outbound request as well as specify an associated routing directive. The routing directive provides a means for a feature to inform the application router how to handle subsequent feature invocation based on the addresses specified in the outbound request.

4. The application selection state associated with the request. This is usually a data structure indicating what the most recently invoked feature was. For a subscription region with more than one subscribed to feature, this information is used by the application router to choose the next feature.

An AR is also free to use any other information when making routing decisions, such as subscriber provisioning data, time of day, network conditions, and so on.

2.2.2 Feature Interaction

An important issue related to telecom feature composition is feature interaction management. Feature interaction naturally arises as a result of composing modular features [30]. In an effort to better manage feature interaction SIP servlet application routing composes features using the “pipes and filters” architectural pattern. Using this approach a signaling chain is formed over composed features: features in a chain are able to absorb, pass through or generate signals upstream or downstream in order to affect overall call state. This form of interaction between features can be exploited to advantage whereby simple features are composed to form complex services. On the other hand, naively composing services can result in unintended interactions between components.

As an example of “good” feature interaction, consider the following example where a callee subscribes to a “Voicemail” (VM) and “Do Not Disturb” (DND) features. The VM feature functions as follows: it behaves transparently in the chain unless it receives a failure message from downstream in which case it switches the caller to a Voicemail recording resource. A simple version of a DND feature functions as follows: if the feature is enabled by the subscriber (e.g., during a given time of day), then when it receives an initial request it unconditionally sends a failure message upstream in response. Otherwise it acts transparently. As shown in Figure 4 these features can be composed so to achieve a desirable feature interaction whereby the failure message emanating downstream from the enabled DND is received by VM which allows the caller to leave voicemail.

Now consider a minor variation on the previous example which highlights the possibility of “bad” feature interaction. In this example the callee subscribes to “Record Voicemail” (VM) and “Call Blocking” (CB). The CB feature functions as follows: if the feature receives an upstream initial request from a caller whose address is on the subscriber’s “black list” then the feature responds by sending a failure message downstream, otherwise the feature behaves transparently, passing through the initial request. If the VM and CB features are composed as shown in Figure 5 then a caller rejected by CB will be able to leave (potentially unwanted) voicemail via VM. While this particular problem can be solved by simply reversing the order that the two features are inserted into the call chain, the more important thing to note is that interactions amongst telecom features need to be managed both to achieve good feature interactions and to avoid bad feature interactions. The pipes and filters architecture adopted by the SIP Servlet specification helps considerably in this regard by isolating feature interdependencies to the signaling connections between features and messages exchanged over those connections. Furthermore, an application router should support the specification and enforcement of an invocation ordering constraint over features. The ordering over features is defined so as to maximize good feature interactions and to eliminate bad feature interactions. An example of such an application router is the DFC application router for SIP servlets which is based on the DFC router [9].

2.3 Comparison of the Domains

To those in the telecom field, telecom application routing may appear to be starkly different from web services orchestration. Those in the web services field may feel that the existing approach to web services orchestration is perfectly applicable to telecom application routing. However, if any unified approach to composition is to be developed it is nec-
necessary to better understand the similarities and differences between the two approaches. To do this we will first discuss the different characteristics of the two domains which lead to different approaches to composition. Then we will compare the representative approaches in each domain: web services orchestration by WS-BPEL, and telecom feature composition by the SIP Servlet Application Router.

2.3.1 Consumer–Provider vs Symmetric Peers
A fundamental difference between the two domains is that in the web domain, a user initiates a service request, for example from a web browser. Although this may result in a web service sending requests to other web services, ultimately all services in the composition are invoked to serve the user. In contrast, in a telecom session there are multiple parties in peer-to-peer relationship. Each of these parties may have applications to serve their needs. Furthermore, the parties may make competing requests at the same time. Therefore the notions of subscriber identity and routing regions become necessary. Management of feature interaction also becomes important.

2.3.2 Explicit vs Implicit Module Invocations
The web services domain is service-centric, i.e. service requests are explicitly targeted to a specific service to be executed. This service in turn may invoke other services, but it looks like a single service from the caller’s point of view. In contrast, when establishing a communication session between multiple parties traditionally no services are explicitly specified. The request only states that a communication session between given users should be established. This view is user (endpoint) centric and entails no notion of a service as such. Modules/services are invoked implicitly during this session establishment process, only if it is discovered that session participants are subscribed to special modules to serve their needs.

2.3.3 Distinct vs Overlapped Functionality
Web services are usually chosen by the composition mechanism to perform distinct functions, for example the steps of checking customer credit and accessing stock in the business process discussed in Section 2.1.1 can be implemented as distinct web services. In contrast, in communications many applications may act upon the same conditions. For example, call coverage applications such as call waiting, record voicemail, and call forwarding are all triggered when the next subscriber feature in the sequence cannot be invoked until the previously invoked feature issues an initial request to the application router. Another distinct difference between web services orchestration and application routing is that composed telecom features are free to exchange subsequent SIP messages (e.g. responses to initial requests or mid-dialog requests and responses) with one another directly via the SIP signaling channel that is established between them. The application router is not involved in this signaling.

2.3.4 Non-Functional Requirements
Non-functional requirements are operational system requirements distinct from behavioral (functional) requirements. For the purposes of comparing the two approaches to composition, the non-functional requirement of interest is the composer response time. Web services composition typically has best-effort response time requirements whereas telecom feature composition has numerous soft real-time requirements as evidenced by the numerous timing requirements present in the SIP protocol standard. The reason for this is that telecom service response times have historically had soft real-time behavior. For example, post-dial delay (the duration that extends between the end of dialing and receiving a call progress tone) is typically bounded. As a result of this difference, the technologies underlying the two approaches to composition differ. Web services composition is supported by what is viewed to be the “heavyweight” WS standards which involve multiple layers of abstraction being traversed between a composer and an invoked web service. In order to minimize post-dial delay, telecom feature composition is relatively lightweight in comparison. The SIP Servlet standard specifies that modules need only be invoked via Java method calls.

2.3.5 Differences in Protocol
In the web services domain, modules are API-based and have the freedom to expose an API that matches the functionality of the service in the best way. In particular, they can select the most suitable name, parameter types and return value types for their APIs, as well as a message format and transport protocol to be used (e.g. SOAP over HTTP). APIs can be also formally described using standardized methods (e.g. WSDL).

In contrast, telecommunication applications are based on protocols (usually SIP) and therefore from the developers’ point of view have all exactly the same interface reflecting SIP features (e.g. SIP servlet API as defined by JSR289). This interface does not reflect in any form their actual functionality, as it is the case for web services. Moreover, there is a lack of any formal specification similar to WSDL that would describe the interfaces of such services.

2.3.6 Interaction Pattern
A web services orchestrator can invoke a module either synchronously or asynchronously, depending on the duration of the task performed by the module. In either case, the orchestrator can be viewed as maintaining an interaction relationship with its invoked modules for the duration of their respective invocations; the modules themselves may or may not interact with one another. A web services orchestrator is free to invoke modules in sequence or in parallel in an order that best satisfies the requirements of the implemented application.

In contrast, a telecom application router only invokes its modules asynchronously. Subsequent to their invocation, no interaction relationship is maintained between an application router and its invoked features. When composing features for a given call, an application router is constrained to invoke the subscriber’s features in sequence. This means that the next subscriber feature in the sequence cannot be invoked until the previously invoked feature issues an initial request to the application router. Another distinct difference between web services orchestration and application routing is that composed telecom features are free to exchange subsequent SIP messages (e.g. responses to initial requests or mid-dialog requests and responses) with one another directly via the SIP signaling channel that is established between them. The application router is not involved in this signaling.

2.3.7 Structure of Composition
Web services should be loosely coupled, and therefore should not have dependencies on each other. Instead they interact only with the orchestrator, for example by means of returning a value back to the orchestrator. The orchestrator may then, in turn, pass the value as a parameter to other
modules involved in the orchestration. Therefore, the execution of the orchestration lies within a single WS-BPEL process (as illustrated in Figure 2). At the same time, the web services that the first WS-BPEL process invokes may in turn be composed by smaller web services. Therefore, the overall orchestration can be seen as a hierarchical tree structure, with the orchestration process at each level under the control of a single business entity.

In contrast, telecom applications are composed in a chain structure (possibly with branches and joins). This chain represents an end-to-end signal flow between telecommunication session participants (as illustrated in Figure 3). Each application in the chain has a persistent connection, i.e. a SIP dialog, with its immediate neighbor (though it should not be aware of its neighbor’s function). This is motivated by the soft real-time requirement in this domain as it reduces the need for a composing entity after session establishment and the need to maintain excessive of state in the composer.

The application chain may traverse multiple administrative domains, for example the caller and callee may be subscribers of different service providers, and thus different segments of the chain (containing their respective applications) will be composed by different composers.

2.3.8 Request–Response vs Session-based

Web service orchestration is not restricted to a chain-like structure, therefore support for parallel processing is possible. Service requests may be received and processed by participants of the orchestration in any order or even in parallel as long as explicit execution ordering dependencies are fulfilled. Therefore, web services orchestration can be described as request-response-driven.

In the telecommunications domain, signaling for a given session stemming from the same origin (i.e. requests or responses) is processed strictly sequentially by applications on the SIP chain, starting with the first (last) element of the chain. Responses are processed by applications on the chain in the reverse order, compared to requests. By design, chains represent a very sequential structure and correct ordering of modules is very important for managing feature interaction.

2.3.9 Inverse State Relationship

A web services orchestrator normally maintains state across module invocations whereas a module it invokes normally doesn’t maintain state after returning from its invocation. In this way a module can be seen to exist only for the duration of its interaction with the orchestrator. During the interaction lifetime, modules normally provide data to the orchestrator which can influence which modules the orchestrator invokes next.

In contrast a telecom feature composer maintains little state – normally just the history of modules already invoked for a subscriber in a region. The majority of state is maintained by the modules invoked by the composer. Because a SIP signaling channel exists between long-lived features, a feature’s internal state can be updated in response to messages received on this channel. However, such message exchanges hidden from an application router because it does not have access to this channel.

2.3.10 Module Behavior and Lifetime

As explained above, a web service used in a composition is typically stateless. The lifetime of a composed module begins when it is invoked. A new instance of this service is created for processing the invocation. Once it is finished, it returns a result and releases the instance, which ends the service’s lifetime. During its lifetime, the module is able to process only the service request that has triggered it and it does not provide any means to interact with this instance during its lifetime. This is typical for small stateless modules that are activated for the duration of a single request. Of course a web service may also be exposed by a larger long running state-full system e.g. an enterprise system.

In telecommunications, composed modules are active processes. Their lifetime begins when they are first triggered during session establishment and it ends when the session is over, e.g. their lifetime can last for the duration of a phone call. During this period, these modules are able to receive signaling (requests and responses), process it and produce new signaling. By doing that they also interact with their “neighbor” modules and affect their behavior.

2.3.11 Different but not Irreconcilable

What emerges from this comparison is that the two approaches differ on a small set of important issues. An application router invokes its telecom features in a sequence; invoked features interact with one another via messaging over a SIP signaling channel that the application router is not privy to; and composition response time has soft real-time constraints.

On the other hand, a web services orchestrator and its invoked web modules interact over the lifetime of the modules; messages are exchanged between a module and an orchestrator, not between modules; and composition response time is best-effort.

With an understanding of how these two composition approaches differ in the abstract we can now identify what is required to reconcile them in the abstract:

1. A mechanism that permits a composer to interact with a persistent telecom feature module after having returned from invoking it. This would enable a composer to be informed of state updates that may occur in telecom modules in response to receiving SIP messages along the separate signaling channel. This would also enable a composer to influence both telecom and web modules after their invocation.

2. A lightweight module invocation mechanism that supports soft real-time limits.

3. Support for managing feature interaction that naturally arises amongst composed modules.

4. Support for telecom feature composition-specific parameters like subscriber, subscription region and precedence order.

In the remainder of this paper we will examine a number of existing approaches that attempt to unify the two approaches to composition. None of these approaches completely solves the problem but each one sheds light on important aspects of the problem.

2.4 Example of Unified Composition

Click-to-Dial (C2D) is the canonical example of a converged telephony and web application. It exposes a telephony functionality — in this case connecting two parties
together in a multimedia session — as a web service such that it may be invoked by other web components. In the simplest case, a user may use a browser to directly trigger the HTTP interface of a C2D application to connect the user’s device and a second party. In more complex use cases, C2D may be used by a composite web application and is triggered indirectly.

It is when composition occurs in both the web and telephony domains, and when there are two or more converged applications, that additional complications arise. In such cases, there are multiple interaction points between the functional modules (i.e., web services and telecom features), thus the techniques discussed in Section 2.1 and 2.2 that can handle composition in their respective domains do not fulfill the requirements of unified composition satisfactorily. As a concrete example, consider two converged applications:

**Talk-To-Agent (TTA)** This is a variation of C2D that may be offered by a company to let a customer request to talk to an agent. When the customer clicks on a button on the company’s web site, a composition of web services performs tasks such as collecting customer profile and selecting the most suitable and available agent. Eventually C2D is invoked to send out SIP messages to call and connect the agent and the customer.

**Do-Not-Disturb (DND)** This is a common application that helps end-users manage their incoming calls. The user can use a web interface to configure criteria for blocking calls, such as time of day and the identity of the caller. When an incoming call arrives, DND may invoke other web services to map a calling number to a name, and then check the name against a black list and a white list to determine whether to allow the call through.

A user Alice subscribes to DND and has configured it to reject all calls to avoid disruption while she is working on an important task. But as part of this task she has to talk to an agent at Acme Inc. Alice clicks on the Talk-To-Agent button on Acme Inc’s web site, but because DND rejects all calls she never receives the call.

In this example, the challenge to a unified composition scheme is to recognize the undesirable feature interaction, and then to take corrective action to provide a desirable user experience. Note that this use case is further complicated by the fact that the two applications are subscribed to by different subscribers. Figure 6 shows TTA and DND composed by two independent web service orchestration processes. They may even operate in two distinct administrative domains. These deployment scenarios offer varying level of difficulties to a unified composition scheme.

### 3. EXISTING APPROACHES FOR CONVERGED SERVICES

In this section we provide a survey of all approaches for converged web and telecom services the authors are aware of. The first three approaches discussed in this section only address the problem of building individual converged applications. The last approach is more comprehensive and begins to address unified composition for converged services.

3.1 Parlay/OSA and Parlay X APIs

These APIs are developed by the Parlay Group to expose the functionality of telecommunication networks to IT applications. They provide a high-level of abstraction that is independent of the signaling protocols used in the underlying networks. In comparison to the more complex Parlay/OSA API [20], the Parlay X API [21] specifies a set of simpler web services specified in WSDL and is particularly targeted for converged web and telephony applications. To use the example in Figure 6, the Talk-to-Agent application can use the Third Party Call functionality to instruct the telephony network to establish a call between two parties. The Do-Not-Disturb application can use the Call Notification functionality to receive notification of calls received by an address, and then to instruct the network to reject a call. The standardization of the web services interface aims to ensure that these applications may be portable across different service provider networks.

There are two issues with the Parlay X API. First, it does not offer any extension mechanism and thus any new telephony functionality must go through a lengthy standardization process, or be exposed using proprietary API. Secondly, the Parlay Group has not addressed the issue of application composition. It is not specified how multiple Parlay X applications would interact if they attempt to control the same call in the telecommunication network. For example, what is the correct behavior if two applications are both notified of an incoming call, but they request opposing call dispositions? A solution has been proposed in [17], where rule-based feature interaction managers mediate Parlay X requests from multiple requestors. However, this is only a partial solution as web service composition is not taken into consideration, and may not be dynamic enough to handle more changing conditions and subscriptions.

3.2 Web Services Initiation Protocol

Instead of maintaining two separate protocol pathways for web services invocation and VoIP signaling, Chou et al. [10] proposed the Web Services Initiation Protocol (WIP)
as a web services replacement for SIP. A WIP endpoint appears as a web service, and the caller may discover and place a call to the endpoint using standard web services mechanisms. Subsequent signaling exchanges between the two endpoints, for example to renegotiate media or to terminate the session, are also performed by web services interactions. The authors claimed better integration with the web and smaller endpoint footprint as the main advantages, and also showed how the WIP endpoint can be extended to perform advanced switching and call control functions. However, it is not apparent how WIP can help manage multiple applications making competing requests for services, and as such this work does not address the application composition problem.

3.3 E4SS Convergence Framework

E4SS (ECharts for SIP Servlets) [24] is a framework for rapidly developing SIP servlets. It provides the developer with the means to specify a SIP servlet as an ECharts state machine, a form of hierarchical state machine derived from the UML Statecharts [3]. The E4SS framework also includes a convergence framework which provides abstractions that make it easy for an E4SS SIP servlet and a non-SIP entity, such as a web service or database, to communicate with one another [4]. The framework provides abstractions for two common types of interaction between a feature and its non-SIP environment:

1. Interactions initiated by the feature such as reading provisioned data from a database or notifying the environment of service status. These interactions are called SIP-to-Java interactions.

2. Interactions initiated by non-SIP components such as initiating the creation of the feature in the first place, or exerting control over its behavior. These interactions are called Java-to-SIP interactions.

As shown in Figure 7, for a given feature the two interaction types can be defined in terms of two interfaces: (1) method signatures comprising its SIP-To-Java interface and (2) method signatures comprising its Java-To-SIP interface. Both interfaces are defined by the feature itself since the decision of how a feature interacts with its environment affects the feature's internal design.

The implementation of a feature’s Java-To-SIP interface methods is clearly the responsibility of the feature itself. This is because these methods will most likely access or manipulate the feature’s internal state. On the other hand, the implementation of a feature's SIP-To-Java interface is clearly the responsibility of the external non-SIP components that interact with the feature. For example, when a feature calls a SIP-To-Java method to indicate its own status, the method will most likely update the state of some external component. Because the SIP-To-Java interface implementation is not the responsibility of the feature, the identity of the external component and the nature of the interaction with the external component need not be known by the feature although these will be known by the SIP-To-Java interface implementation.

To date the E4SS convergence framework has been used extensively to support interaction between telecom features and non-SIP entities such as web applications and databases. For example, consider the DND (Do Not Disturb) feature described in Section 2.4. This feature may define a SIP-to-Java interface that includes a method that will be called by the feature to determine if an incoming call should be rejected. The implementation of this method could consult provisioned data for the callee to make its determination. The framework also supports the ability to reuse telecom features: the customizable aspects of a feature are specified in terms of its SIP-to-Java and Java-to-SIP interfaces. By providing SIP-to-Java implementation and calling the Java-to-SIP interface appropriately, a feature’s interaction with its SIP and non-SIP environment can be customized without requiring modifications to the feature itself.

3.4 Composition Engine

Ericsson Composition Engine (ECE) [11,15] is a system for composition of converged application services from web, enterprise and telecommunications domains, both circuit switched (CAP/INAP) and IP (SIP) based, able to control feature interactions across technology borders.

The core of the engine is technology and protocol agnostic; support for new technologies can be added with low effort. Currently supported technologies and protocols includes web services and REST-based services typically used by Web 2.0 applications on the Internet and in the enterprise domain, as well as SIP in an IMS context and CAP/INAP in an IN context. Also services available in ESBs are supported through JBI. Any of these protocols can be used to implement constituent modules used for the creation of composite converged applications and for triggering of such composite applications. The set of supported protocols also permit using the ECE for SCIM and IM-SSF nodes as defined by 3GPP.

The ECE performs composition of services on the macro-level according to the SOA approach and can compose heterogeneous modules (see Section 2.1) [12]. It employs an application router towards execution of SIP features and a web services orchestrator for execution of web services thereby enabling composition of converged web and telecommunication applications. Composite applications are created as “application skeletons”, designed as a model of the core business-logic of the application in terms of participating modules. Protocol-level details related to the interaction with modules are left to the Composition Execution Agents.
compose SIP applications (possibly across different SIP servlet containers and sessions) taking into account not only SIP-specific information, but also being influenced by non-SIP modules participating in the composition. For instance, routing at the SIP level may depend on address information dynamically obtained from a corporate Microsoft Exchange address book by means of a WS invocation. Similarly, non-SIP applications can be invoked by the ECE based on SIP signaling or status of SIP features.

- receive a callback from composed modules not only on initial signaling while a session is being set up, but also on subsequent signaling after session establishment. These callbacks may have no influence on the composition structure of the established SIP session, but may trigger composition activities in non-SIP domains. Aspects of the composite service may continue to execute even after SIP session establishment.

- handle composition sessions, which may include multiple telecommunication and web sessions.

- enable data exchange between modules involved in compositions, thus sharing data between different technology domains.

The ECE employs constraint satisfaction mechanisms for feature interaction control. This requires skeletons to contain dependencies between constituent modules. Such dependencies are expressed by means of constraints. Concrete services instances are selected for invocation after fulfilling all related constraints. This ensures that the resulting composition is “correct by construction” as long as the set of constraints is complete and correctly models the application domain area.

To summarize, the ECE provides higher-level abstraction mechanisms for the composition of heterogeneous applications. This bridges many of the technological differences between web and telecommunication applications and provides a unified way for creating converged composite applications. In contrast, SIP applications deal with low-level specifics of the SIP protocol, which are complementary to the high-level logic executed by the ECE.

4. FUTURE DIRECTIONS

Having defined the solution space for the unified composition problem we now consider a number of promising research directions that aim to solve the problems identified. Specifically we merge two prominent approaches to introduce a new unified mechanism that combines the advantages of both web and telecom type composition. This composition mechanism maintains control of the business logic of the application and all its state, and can communicate with all modules at any time. It can execute modules across domains, sequentially and in parallel, while taking into account feature interaction and maintaining soft real-time performance.

4.1 Unified Composition Framework

We began an investigation into a Unified Composition Framework (UCF) by combining the E4SS Convergence Framework and the Ericsson Composition Engine (ECE), discussed separately in Sections 3.3 and 3.4 above. The initial investigation focuses on telecom modules developed as SIP Servlet applications.

In the original ECE approach there is no way for the ECE to discern state changes occurring in individual SIP applications. For this reason the original ECE inserted helper applications around each SIP feature on the SIP chain in order to intercept all low-level SIP signaling messages and to report them to the ECE such that the composition execution (skeleton) may take action (e.g. invocation of a Web Service or invocation of a SIP feature) on certain signals. This increases the processing load and latency in the flow of SIP signals along the application chain, and also introduces a potential bottleneck in the ECE. There is also no standardized mechanism for the ECE to send commands to the SIP applications, particularly after communication session is established.

More importantly, this approach requires the writers of composition skeletons to handle the low-level SIP requests and responses. The ability to express the composite logic in high-level terms is crucial for the creation of easy-to-develop, clean, compact and highly reusable compositions. Low-level handling of protocol and technology-specific details should be avoided if possible and hidden in the implementations of features used in compositions.

In the UCF approach, we use the convergence support in E4SS to expose high level, application-specific interfaces to the SIP applications. This is shown in Figure 8. The ECE no longer receives all SIP messages into and out of each SIP application. Instead, once the application chain is established, the applications only notify the ECE of events relevant to the application feature logic. For example, a DND applica-
tion may notify the ECE via its SIP-to-Java interface that it is rejecting a call for certain reason. (In case of legacy SIP applications that are developed without using the Unified Composition Framework, low-level interception of SIP signaling around such applications may be still necessary.) The interface may also allow specific command actions from the ECE. For example, the ECE may instruct an application to play a prompt because a high-priority email to the user has been received.

The interface to each SIP application may be standardized as a Web service interface, for example using WSDL and SOAP over HTTP. However, the actual events and commands that each application provides are application-specific, and thus the UCF is extensible and will not restrict innovations in telecom features. This approach is also amenable to automatic tooling support. For example, the developer of a SIP application may write a Java interface class to specify the set of commands and events it supports. A tool can then automatically generate the appropriate WSDL and codes to implement the SOAP/HTTP interface.

This approach removes the ECE from the SIP signaling path (except in the initial setup of the application chain). It also isolates the low-level SIP signaling messages from the composition execution. This is beneficial both in terms of ease of programming for the composition author, and encapsulation and separation of concern — the propagation, interception and interpretation of SIP messages are confined to the SIP applications layer.

Figure 9 also shows a new unified API to the ECE. Besides support for event notifications from SIP applications, web services modules may also use the same API to interact with the ECE. It is beneficial to standardize this API such that applications and composition engine implementations may be portable.

For feature interaction management, in the telecom domain the DFC AR (Section 2.2.2) makes use of user subscription, routing regions, precedence relationship among applications, and routing directives to construct an ordered application chain and to manage feature interaction. At the same time, the ECE uses constraints satisfaction mechanisms to manage dependencies and relations between modules (Section 5.4). For the UCF approach, we are investigating incorporating both approaches into a unified composer function.

4.1 Example Solutions

To illustrate the concept of the UCF, we return to the example problem in Section 2.3. Figure 10 shows a deployment where a single Unified Composer is responsible for all Web and SIP services of both Acme and Alice. Alice’s request to talk to an agent is received by the composer. After invoking a few Web services to bring up Alice’s customer profile and select an available agent, the composer invokes C2D. When C2D calls Alice, the composer is queried to select the next SIP application. In this example, the composer selects DND as in the normal case. However, when DND notifies the composer of an incoming call and asks whether to block the call, the composer responds that the call should be allowed through. This is equivalent to adding Acme to the white list for just one call.

Figure 11 shows a more realistic deployment where Acme and Alice have different composers. It is necessary that Alice’s composer is aware that Alice is making the original request. In this scenario, Alice’s request is handled by her composer, which in turn invoke Acme’s composer to request the talk-to-agent feature. When C2D calls Alice, Alice’s composer is queried. The rest may proceed as above. However, to show an alternative, the composer may elect to skip DND altogether and select the next application. This has the same effect of letting the call from Acme through despite Alice’s DND setting.

4.2 Explicit Telecom Module Invocation

In Section 2.3.2 the current approach to telecom module invocation was described as being implicit. This is because the initial request received by an application router from the environment does not explicitly identify a module to invoke on behalf of the caller or callee, rather, the caller and callee’s modules are invoked as a side-effect of establishing an end-to-end session between the endpoints specified in the initial request. However, if we choose to use a different interpretation of the addresses in an initial request, it is possible to re-interpret implicit invocation as explicit invocation. In particular, one can view the caller and callee address in an initial request as module identifiers instead of endpoint identifiers. This way different caller and callee modules can be invoked based on the caller and callee address values.

If caller and callee addresses are now used to identify modules, this raises the obvious question of how the caller and callee endpoints themselves are now identified. For a calllee,
the answer is that the last module in the chain of modules for a caller is responsible for forwarding the initial request to the address of the callee’s physical endpoint. For a caller, the answer is that the first module subscribed to by the caller’s physical endpoint is responsible for forwarding the initial request so that it appears to come from a callee address that differs from the physical address.

As an example, consider Figure 11 from the previous section which outlines a possible solution to the example problem. If we assume that the top-level composed service supplies the underlying C2D service with a callback address for Alice that differs from the address of her physical endpoint, then this callback address can serve to identify a callee service for Alice that intentionally excludes Alice’s DND service.

To generalize this approach, a caller or callee can maintain many different addresses that are not associated with physical endpoints. Each such “virtual” address can identify different modules constituting the composite service for its associated address. By utilizing modules that forward from one caller or callee mobile address to another, it is possible to invoke different composite service sets on behalf of a caller or callee. It is important to realize that a caller or a callee “owns” their virtual addresses in the same way that they own the address of record associated with their physical endpoint. In this sense, a user’s virtual address subscribes to the modules they are identified with in the usual way.

This approach to using addressing to identify service sets was introduced in the context of address hierarchies in DFC [29]. This work proves a number of interesting theorems that emerge when one defines an abstraction ordering on the set of addresses. As an example, this work shows that forwarding between virtual addresses representing a caller should proceed from less abstract (e.g., device-specific) to more abstract (e.g., representing a caller’s role) in order to maintain a caller’s privacy. Similarly, for a callee, forwarding should proceed from more abstract to more concrete virtual addresses. There are also initiatives underway in the IETF [23] and the 3GPP IMS [2] to identify services with initial SIP requests. Instead of utilizing the existing SIP address information these approaches involve the use of a “service identification” header in an initial request to identify the desired services. Future work is required to assess the tradeoffs associated with these different approaches.

5. CONCLUSIONS

In this paper, we have examined and contrasted in detail composition in the SOA and Web application domain with composition in the telecom and SIP domain. We also discussed the motivations for developing techniques for unified composition of telephony and Web services. A survey of existing approaches was presented, following which we discussed a number of promising future directions. We hope that the contributions in this paper may help motivate and guide further research in this important area.

6. ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of Hal Purdy, Tom Smith, Venkita Subramonian and Pamela Zave at AT&T, and Eugen Freiter, Göran AP Eriksson, Patrik Wiss, Kristoffer Gronowski and Per Pettersson at Ericsson.

7. REFERENCES


