SewNet - A Framework for Creating Services Utilizing Telecom Functionality

Sumit Mittal, Dipanjan Chakraborty, Sunil Goyal, and Sougata Mukherjea
IBM India Research Laboratory
New Delhi, India - 110 070
sumittal@in.ibm.com, cdipanjan@in.ibm.com, gsunil@in.ibm.com,
smukherj@in.ibm.com

ABSTRACT

With Telecom market reaching saturation in many geographies and revenues from voice calls decreasing, Telecom operators are trying to identify new sources of revenue. For this purpose, these operators can take advantage of their core functionalities like Location, Call Control, etc. by exposing them as services to be composed by developers with third party offerings available over the Web. To hide the complexity of underlying Telecom protocols from application developers, the operators are steadily adopting Service Oriented Architecture (SOA) and reference standards like *Parlay-X* and *IMS*. However, a number of challenges still remain in rapid utilization of Telecom functionalities for creating new applications - existence of multiple protocols, different classes of developers, and the need to coordinate and manage usage of these functionalities. In this paper, we present SewNet, a framework for creating applications exploiting Telecom functionality exposed over a (converged) IP network. More specifically, SewNet a) provides an abstraction model for encapsulating invocation, coordination and enrichment of the Telecom functionalities, b) renders a service creation environment on top of this model, and c) caters to various different categories of developers. With the help of two use-case scenarios, we demonstrate how SewNet can create services utilizing rich Telecom functionality.

Categories and Subject Descriptors

C.2.m [**Computer-Communication Networks**]: Miscellaneous; D.2.6 [**Software Engineering**]: Programming Environments

General Terms

Design, Languages

Keywords

Service Composition, Telecom, Web 2.0, Abstraction Model

1. INTRODUCTION

The Telecom business model is evolving. With the market reaching saturation and revenues from voice calls decreasing rapidly, Telecom operators are aggressively looking at newer sources of revenue. This includes partnerships with third-party providers to offer alternate services such as gaming applications, news, ringtones, etc. In recent years, however, these services are facing strong competition from *similar* technologies and applications provided by Internet Content providers. These applications can be accessed through a browser-enabled phone, while paying only for the connectivity charges, and thereby adversely affect revenues from the paid-for-services hosted on the Telecom operator portal. Examples of such services range from VoIP and telephony conferencing services to various content services (maps, ringtones, etc.). An increasing number of mobile users are now using browser-enabled phones to access these services, bypassing the Telecom portal. For example, it has been estimated that 60% of the mobile content traffic in US and 90% in Europe is off-portal [10].

Telecom operators, however, have an edge over Internet service providers in terms of their still unmatched core functionalities of Location, Presence, Call Control, etc., characterized further by carrier-grade Quality-of-Service (QoS) and high availability. Therefore, a potential channel for the operators to increase their revenue is to offer these functionalities as services to developers for creating new innovative applications. These developers can belong to not only the select partners of the Telecom operator, but also those involved in creating a variety of long–tail applications [18]. Moreover, with the underlying IP and telephony networks converging, developers can compose these functionalities with third party services available on the IP network. For example, Location and Presence information from Telecom can be clubbed with Google Maps to provide new workforce management solutions for mobile settings [15].

To support the basic operations such as voice and SMS, many of the building blocks of a Telecom infrastructure - location registries (HLR/VLR), accounting and billing services, etc. - are already in place. However, these are not easy to utilize in new applications because they are not exposed using standardized frameworks and component models. Towards this, Telecom operators are steadily adopting Service-Oriented Architecture (SOA) that would let developers access these services without knowledge of the underlying platform implementation. Web services, as an instantiation of SOA, have received much interest in the community due to their potential in facilitating seamless business-to-business or enterprise application integration [22]. The Parlay consortium has defined a standard, called *Parlay-X* [8], that exposes Web service interface for core Telecom functionali-

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

ties. On similar lines, IP Multimedia Subsystem (IMS) [4] provides a reference framework to expose these functionalities as services to Web-engineered systems using SIP [19].

Although efforts like Parlay-X and IMS are a step in the right direction, rapid development of applications that utilize Telecom functionality still faces a number of challenges in a realistic setting. Firstly, one needs to provide interfaces that shield the application developer from different Telecom protocols (Parlay-X, SIP, etc.), including the legacy ones. Secondly, one needs to package the Telecom functionalities so that they can be *readily* used in different programming styles (Java, HTML/JavaScript, etc.) other than pure Web service based composition (like BPEL). Finally, one also needs to encapsulate the invocation of Telecom functionality with various *coordination* rules, for example, those that correspond to managing the usage of a service, including monitoring, metering and access control.

In this paper, we present SewNet, a framework that addresses the above challenges to enable rapid composition of Telecom services. Our main contributions in SewNet can be summarized as:

- We propose the Telecom Service Reference, Encapsulation and Coordination (T-Rec) 'Proxy' model. This enables developers to seamlessly incorporate Telecom functionality and apply various coordination rules.
- We design a service composition environment based on the T-Rec model, and implement a prototype.
- We demonstrate how SewNet can be used by different categories of developers, including Java, BPEL and HTML/JavaScript programmers.

It is important to note that although we focus on Telecom in this paper, the T-Rec proxy model is generic and applicable to third-party services available on the Web.

The rest of this paper is organized as follows. In Section 2, we motivate the reader towards challenges and issues in composing Telecom functionality in different applications. We then propose and describe our T-Rec model in Section 3, followed by SewNet's architecture in Section 4, and implementation in Section 5. Section 6 illustrates the use of SewNet for two use-case scenarios, while Section 7 provides a discussion of related work. We conclude in Section 8.

2. PROBLEM ILLUSTRATION AND MOTIVATION

We illustrate the problem with respect to the component-oriented diagram of a service that utilizes Telecom functionality, as shown in Figure 1. In general, such an application can be broken into two major blocks. Firstly, there are *Telecom blocks* (represented as red rectangular boxes in the figure) that invoke a Telecom network functionality (for example, invoking the location service or capabilities like SMS, Third Party Call Control). The others are *non-Telecom blocks*, where the developers can embed various constructs (depicted by green hexagonal boxes in the figure). For example, in a workforce management solution, these blocks can contain logic for scheduling agents on the basis of Location and Presence information provided by Telecom operator. Alternatively, these blocks can be UI constructs, for instance those enabled by various Ajax-based platforms. Finally, the non-Telecom blocks can also be invocation points

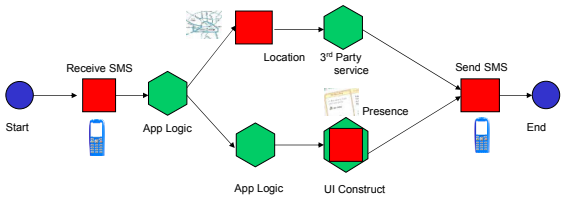

Figure 1: Model of a Telecom Service

for orchestration with third party services available over the Web. Also, as the figure shows, to bind these Telecom and non-Telecom blocks, specification of complex control and data flows is also required during the service design.

We believe that for application development utilizing Telecom blocks composed with the non-Telecom ones, the following challenges need to be addressed:

Firstly, functionality available within a Telecom operator is, in general, exposed using multiple protocols. For example, Presence related information can be accessed via the SIP protocol, and Messaging capabilities using SMPP protocol. As a step towards hiding protocol heterogeneity and complexity, the Parlay consortium has come up with the Parlay [9] and the subsequent Parlay-X standards [8]. Parlay-X exposes a Web Services interface for several Telecom functionalities. However, it does not cover the whole gamut of functionalities that can be offered by the Telecom operator, especially those requiring session control. Further, some of the Telecom functionality can also be exposed through legacy protocols. Therefore, we need an *abstraction* model that provides interfaces shielding the application developer from the underlying protocols. This model should also allow seamless switching between different protocols, for example, when moving from legacy interfaces to the Parlay-X ones.

Secondly, developers who want to utilize Telecom functionality in their application can belong to different categories [18]. More specifically, composite applications modeled in Figure 1 can be written in Java, BPEL, HTML/JavaScript, etc. Although editors corresponding to the various programming styles provide the developer with constructs for the non-Telecom blocks, they still require the Telecom functionalities packaged in a format suitable for incorporation. For example, in the case of Java applications, a developer needs a Java interface to invoke these functionalities (while a Java-based programming environment lets her code much of the non-Telecom blocks). Similarly, developers require a WSDL interface for a BPEL-based composition, JavaScript for a HTML/JavaScript based composition, etc. Therefore, the abstraction model (outlined above) for core Telecom functionalities needs also to be *broad* to cover a range of programming styles.

Thirdly, even though interfaces like WSDL (for Parlay-X) and SIP have tools to generate "clients" for invoking the corresponding functionalities, in real life, however, there is effort required to integrate these clients within the application. For example, code needs to be written to incorporate the client in the application code, while taking care of tertiary library dependencies for this client. It would help the developer immensely if the abstraction model pre-generates the clients corresponding to different programming styles, and packages them in a *structured* manner. Having a well-defined structured format would enable any application de-

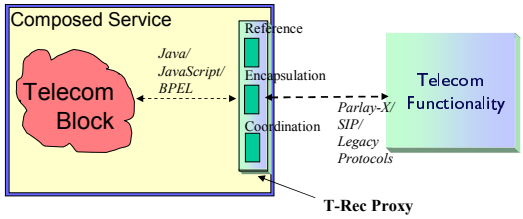

Figure 2: T-Rec 'Proxy' Model

velopment environment (with some extensions to interpret this structure) to integrate these clients seamlessly.

Finally, when Telecom functionality gets used in an application, Telecom operator as well as the application developer want to coordinate its usage. For instance, one needs mechanisms for embedding logic for charging, specifying access control policies, etc. Furthermore, with recent trends suggested by Web 2.0, application developers should be able to contribute, implicitly or explicitly, to the enrichment and refinement of the exposed Telecom functionality (and its usage). Therefore, the abstraction model needs to be *rich* enough to enable all of this.

A number of operators are already moving in the direction of making their core functionalities available for application development. For example, British Telecom has released a Software Development Kit [11] that enables its network services to be utilized in Web mash-ups. However, what is missing is an abstraction model which is broad, structured and rich, as motivated above. We propose such a model next, and thereafter describe a service creation framework on top of this model.

3. T-REC MODEL

Figure 2 represents the basic concept of our Telecom Service Reference, Encapsulation and Coordination (T-Rec) 'Proxy' model. In essence, once an application has been broken down into Telecom and non-Telecom blocks, this model is used to realize the Telecom blocks, considering the programming style, while also enabling mechanisms for coordination and enrichment. In practice, these proxies would be created by a Telecom operator and made available to application developers over the Web (or a converged IP network).

3.1 T-Rec Structure

Our rich, structured T-Rec Proxies consist of the following elements:

- *Proxy Representation.* Contains signatures of the methods (APIs) exposed by the proxy along with a textual description of the service it represents. The APIs are designed to hide protocol specific details and abstract the Telecom functionality to the programming language level. As discussed before, APIs corresponding to multiple styles (Java, BPEL, JavaScript) should be created to support different environments.

- *Implementation.* This module connects to the Telecom service using the underlying protocol, and is available in different formats. For example, the implementation could be in the form of a *.jar* file for a Java proxy, a *.js* file for JavaScript, or could be encapsulated by visual constructs, such as widgets, and used inside HTML pages.

- *Configuration File.* Proxies come with a default setting but can be further configured by developers. This includes assigning default values for some of the parameters in an API, specifying the access control list, etc. Such settings could also be functionality specific, for instance, restricting the size of SMS messages.

- *Metadata.* To enable easy look up, keywords and tags related to the proxy functionality are associated with it. New tags can be added to the proxies if required; for example when a developer utilizes a proxy in a way that was not originally foreseen by its creator.

- *Utility Snippets.* The proxies are populated with multiple code snippets on top of the basic functionality. For example, a 'Presence' proxy may have a program fragment that parses the returned response (usually an XML document) for different attributes. These utilities can be suggested, in an appropriate manner, to developers who wish to use the proxy.

- *Unit Test Code.* Proxies contain codes that let different APIs supported in the proxy be tested in isolation. These are very helpful during testing and debugging.

- *Link to Blogs.* Each proxy is linked to a blog entry where developers can log their experience of using the proxy. If multiple proxies are suggested during a look up, analyzing the blog entries can help the developers choose the most appropriate one for their task.

3.2 T-Rec Benefits

Intuitively, a T-Rec proxy acts as a 'wrapper' for Telecom functionality, including its underlying protocol. Using this wrapper, the proxy creator can provide several benefits to application developers.

3.2.1 Encapsulation

APIs defined in a proxy can hide protocol specific details from the developer. For example, interfaces in Parlay-X throw exceptions with error codes that require knowledge of Parlay-X for interpretation. As an instance, an application developer using Parlay-X would need to know that the error code *SVC0004* stands for invalid addresses in a message. Using the proxy model, we can encapsulate these error codes with higher level exceptions, such as throwing *InvalidAddressException* whenever error code SVC0004 is returned. Moreover, using proxies, similar APIs can be exposed across different protocols. For example, various APIs in the Location proxy can have similar signatures for Parlay-X and SIP-Presence based implementations[1].

3.2.2 Coordination

When Telecom proxies get used in an application, Telecom operator as well as the developer can manage and meter its usage. For instance, whenever the proxy corresponding to Location information gets invoked within an application, the Telecom operator can authenticate the developer and also charge some amount. In this case, proxies are configured to collect the relevant information, for example developerId, from the developer and send to the operator. Similarly, the

[1]In SIP, location information is obtained by subscribing for the presence information, and parsing the returned document. We can wrap this under a *getLocation()* interface.

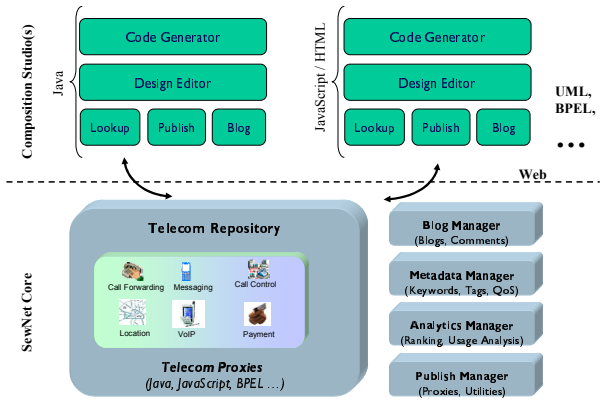

Figure 3: SewNet Architecture

developer can configure the Location proxy to cache the location information locally within itself, and avoid connecting to the operator's infrastructure at each invocation.

The proxy model also provides an easy mechanism to incorporate various business contracts between the operator and the developer. For example, implementation module in a proxy can be extended to make the proxy display advertisements on behalf of the operator, whenever it is invoked. In this case, logic can be such that the proxy picks what to advertise on a real–time basis.

3.2.3 Collaboration and Reuse

Using the proxy model, developers can collaborate, share and contribute towards enriching Telecom functionality. For instance, user of the Location proxy in an application can publish a utility to parse the output of this service. This utility can be re-used by other developers while incorporating this proxy in their applications. Similarly, the proxies can be configured to provide updates to a developer about new entries on the blog, utilities published recently and bug fixes, etc. In the case of bug fixes, logic can be embedded in the proxy to automatically download the latest implementation modules.

4. SEWNET DETAILS

In this section, we present SewNet, a framework that utilizes the rich, structured T-Rec proxy model to enable seamless weaving of Telecom functionality with application logic and other constructs required to develop a service.

4.1 SewNet Architecture

As Figure 3 shows, SewNet has two main architectural components - SewNet Core and Composition Studio(s).

4.1.1 SewNet Core

SewNet Core forms the backbone architecture that exposes Telecom functionality to developers through simple, intuitive interfaces for lookup and select while allowing for developer participation and feedback through publishing and blogging.

Telecom Repository consists of proxies for different functionalities exposed by the Telecom operator. These proxies are available in various implementation styles, for example, Java proxies to be used inside Java applications, JavaScript proxies to run on a Web browser, etc. As mentioned earlier, each proxy hides the underlying protocol (Parlay-X, SIP, etc.) and offers a rich set of APIs to facilitate integration with the application being developed.

Metadata Manager helps the Telecom Repository organize and maintain relevant metadata (keywords and tags, textual description, etc.) associated to a proxy and the APIs it offers. This information is used to suggest proxies on a look up.

Blog Manager organizes and stores free-form textual comments associated to a proxy and its APIs. These inputs are presented to the developer while browsing and selecting proxies from the Telecom repository.

Analytics Manager maintains qualitative information about proxies, including a rating and ranking of each proxy. We envision this manager containing tools to analyze blogs by different developers, collect usage statistics, etc. and making such information available to application developers.

Publish Manager defines the interface to publish new proxies as well as new artifacts associated to an existing proxy; published items become available to other developers.

4.1.2 Composition Studio(s)

Developers wishing to use Telecom proxies exposed by SewNet need to integrate their development environments (or composition studios) with SewNet Core. These studios range from programming platforms (e.g. Eclipse environment) to model driven tools (such as those containing UML editors for service design and representation) to workflow editors allowing services to be composed in a language like BPEL. For integration with SewNet Core, a studio needs extensions along three dimensions. Firstly, its service design (or programming) editor should provide the ability to identify the Telecom blocks from the non-Telecom blocks. Secondly, it should offer Lookup, Publish and Blogging interfaces for proxies provided by SewNet core. Thirdly, once proxies have been selected for different Telecom blocks, a Code Generator module should traverse the structured format of each proxy to seamlessly integrate it with rest of the application code[2]. It is interesting to note that a particular composition studio may use one or more different types of proxies. For example, while creating a JSP page, developers can incorporate Java as well as JavaScript proxies.

4.2 Composing Telecom Services using SewNet

Once a composition studio has been integrated with SewNet Core, the following steps illustrate the process that application developers follow to compose Telecom services:

1. *Service Design*: Developer designs the service using drag-and-drop or other mechanisms supported on the design editor. In this step, Component services, Logic blocks, Control flow (sequencing, fork, join), etc. are defined by the developer (c.f. Figure 1).

2. *Proxy Lookup*: For each Telecom block, developer obtains a set of matching proxies from repository, based on keywords, input-output, or both.

3. *Proxy Selection*: From candidate proxies suggested by SewNet, the developer selects those that best fit the requirement. Proxy selection is based on suitability

[2]In practice, a Composition Studio would already have some code generation capabilities. In this case, we just need to extend these capabilities to incorporate the proxy model.

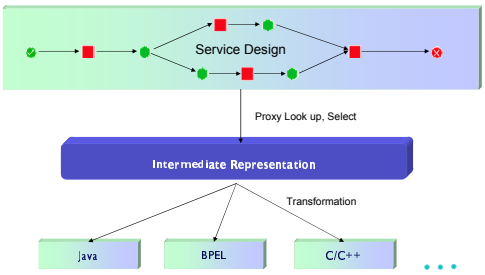

Figure 4: Code Generation

(for e.g. reading more about each proxy) and other metadata (QoS parameters like reputation, etc.).

4. *Proxy Configuration*: Developer optionally associates various service coordination rules with each proxy. For example, she specifies the time-period after which cached location information is to be refreshed by the Location proxy.

5. *Code Generation*: Outputs code for the designed service (BPEL, Java, JavaScript inside HTML, etc.). This step is described in more detail in the next subsection.

6. *Incorporate Other Constructs*: Developer incorporates appropriate application logic, UI elements, etc. to complete the service. At this point, developer also takes care of data-flow between different constructs.

4.3 Code Generation

From a developer's view, once a service has been designed and relevant proxies for the Telecom blocks selected, she expects the composition environment to generate a skeleton code that not only captures the service flow, but also integrates code for the selected proxies. Further, this code should provide her the extension points to include application logic and other constructs for the non-Telecom blocks. We divide this process of 'Code Generation' into three steps:

1. *Capture Service Design:* Service design is captured in a structured document, that we call *processDoc*. Further, for each Telecom block, `processDoc` stores information about the proxy that was selected, and the API that was chosen under this proxy. It should be noted that the structure of `processDoc` and the information it stores is dependent on the programming style of the application. More specifically, it should be able to represent each programming construct of that style.

2. *Generate Process Skeleton:* This step takes `processDoc` and transforms it to a concrete, fully-compiled code. For this purpose, Code Generator parses `processDoc` and converts each element into the corresponding programming construct. While parsing this document, it creates place holders for the non-Telecom blocks and adds comments to aid the developer when she examines the generated code. For the Telecom blocks, it populates the proxy code, as described next.

3. *Populate Telecom Proxy Code:* In this step, the Code Generator imports the relevant implementations of the proxies from the Telecom Repository and generates the code necessary to invoke the selected API in the proxy. The code produced depends on the programming language. For example, while for Java, it creates a *class* with an invoke *method* that internally calls the proxy API, for BPEL, it generates an invoke *statement*. Further, Code Generator also analyzes the method signature of each API to understand the exceptions being thrown and organizes appropriate exception-handling blocks around it. Finally, Unit Test codes available with each proxy are included in the code generated.

5. IMPLEMENTATION

In this section, we describe an implementation of SewNet, including various Telecom proxies, a composition studio to develop services using these proxies, and generation of code in two different programming styles - Java and BPEL.

5.1 Telecom Repository

In our current implementation, we have built T-Rec proxies corresponding to SMS, Location and Presence functionality and are in the process of creating a proxy for Third-party Call Control (3PCC). While SMS, Location and 3PCC are designed on top of Telecom Web Services Server[3] (TWSS), Presence proxy is developed on SIP interfaces exposed by WebSphere Presence Server (WPS)[4]. TWSS is an offering from IBM that enables Telecom operators to provide developers with controlled, reliable access to network capabilities such as Location, SMS and Call Control through standards-based Parlay-X Web Services. On the other hand, WPS is an application that collects, manages, and distributes real-time presence information to applications and users based on the SIP protocol.

Proxies for each functionality were created in three programming styles - Java, WSDL and JavaScript. As noted earlier, each proxy has an implementation module that connects to the service through the underlying protocol, a service representation object (including metadata about the proxy), a module to test the proxy in isolation and finally a set of attached utilities. All proxies are available through a repository implemented on top of DB2 [5]. This repository provides various interfaces for integration with a composition studio - proxy lookup and import, publishing of new proxies and utilities, blogs indexed per proxy per API, etc. We are currently in the process of including the capability to analyze blogs and feedback received for different proxies.

For SMS and Location, BPEL proxies were created by using the WSDL interfaces defined by Parlay-X, these provide sufficient information for invoking the corresponding functionality from a BPEL workflow. On the other hand, Java proxies were implemented by generating Java clients from these WSDL descriptions (using standard IBM tools) and wrapping inside our proxy structure. For Presence proxy based on SIP, we used the JAIN SIP standard and rendered the interfaces in Java for publish, subscribe (with notification handling), etc. This implementation was also exposed as a Web service, using which a WSDL interface was obtained for the BPEL proxy.

For Parlay-X based functionality (SMS and Location), JavaScript proxies can be created by including standard SOAP over HTTP calls from within a JavaScript code frag-

[3]http://www-306.ibm.com/software/pervasive/serviceserver/
[4]www.ibm.com/software/pervasive/presenceserver/
[5]www.ibm.com/db2

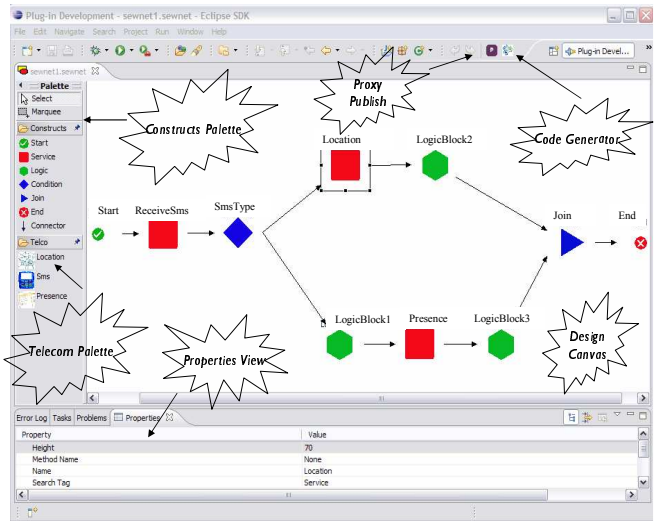

Figure 5: SewNet Composition Studio

ment. However, for SIP based proxies like Presence, the above procedure does not work since SIP messages are exchanged over TCP/UDP. To create a JavaScript proxy in this case, we first implemented a servlet (to be hosted on the operator's infrastructure) that talks to the Presence Server using SIP (over UDP) messages. In turn, the Presence JavaScript proxy interacts with this servlet to fetch presence-related information while shielding the developer from specifics of SIP protocol.

Apart from proxies, our repository contains a Telecom Constructs Library (TCL) to offer developers with *ready-to-use constructs* that can be utilized in a variety of Telecom applications. For example, this library consists of data-type definitions based on the TeleManagement Forum's Shared Information/Data (SID) model[6] for the Telecom industry. TCL also contains definitions of various exceptions related to the Telecom domain, such as *NetworkBusyException*, *UnknownEndPointException*, etc. These exceptions encapsulate various protocol specific error codes, like those defined in the Parlay-X standard (c.f. Section 3.2.1).

5.2 Composition Studio

Figure 5 gives a snapshot of a prototype Composition Studio for SewNet along with an annotation of its different components. The `Constructs palette` offers constructs to help build services, for example, those for specifying start and end blocks, differentiating a Telecom block from application logic block, a condition statement, a join, etc. On the other hand, `Telecom palette` exposes proxies for accessing functionalities of the Telecom network, and currently supports SMS, Location and Presence. The `Design Canvas` lets the developers design their services through simple drag-and-drop of various constructs and Telecom functionality. The `Properties View` displays attributes attached to different components of a service design. Once the service has been completely designed and the proxies chosen, developer can generate code by invoking the `Code Generator`. Currently, code can be generated in Java and BPEL.

The Composition Studio provides service developers the facility to look up existing proxies and also publish new ones. The proxy look up and publish interfaces are shown in Figure 6. The look up is enabled by SewNet's Metadata Manager and can be invoked using simple keywords. The results first display a brief description of the suggested proxies following which the developer can ask for more details for the ones that match her requirements. At this point, developers can also check blogs to benefit from other users' experience. The publish interface, on the other hand, provides an easy mechanism for developers (operators in the case of Telecom functionality) to package their services as a proxy, and publish the same in the repository. These proxies can be looked up and reused by others, thereby fostering a collaborative environment.

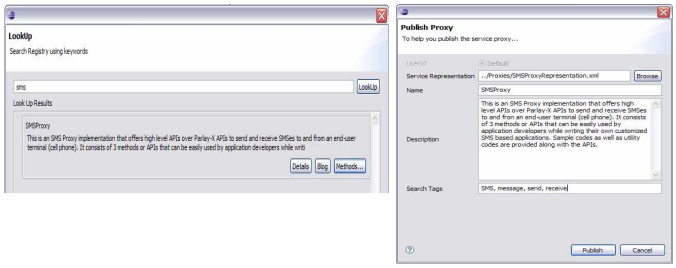

Figure 6: Proxy Look Up and Publish

To guide the developers in their composition process, Composition Studio also includes a set of application templates. These templates range from being simple applications utilizing core Telecom functionality to workforce management solutions and other such services making extensive use of Location and Presence information. Developers using SewNet have the option of starting with these templates, and refining them in the design canvas as per their requirement. Code Generator can then output code for these.

The entire Composition Studio is implemented as a plug-in to the Eclipse platform[7]. Eclipse is an open source *meta* IDE built using Java and can be used for building other IDEs. By becoming part of the Eclipse framework, this studio becomes immediately available to the large community of Eclipse users. Moreover, the plugin based approach allows tapping into various existing and future components being added to the Eclipse platform. For example, generated Java code is directly editable using plug-ins included in Eclipse.

5.3 Service Coordination

As discussed before, developers can coordinate various aspects of their usage of Telecom functionality through the proxy model. The current implementation of these proxies helps developers coordinate their execution (in the developed application) with respect to testing, billing and configuring various application specific requirements.

5.3.1 *Testing and Debugging*

A major challenge for developers in Telecom domain is to perform the tasks of testing and debugging without going 'live' on the network. Towards this, SewNet provides an implementation of proxies for a Telecom simulator[8]. This simulator mocks the basic operations of SMS, Call Control, etc.

[6]http://www.tmforum.org/browse.aspx?catID=1684

[7]http://www.eclipse.org

[8]http://www.openapisolutions.com/

based on a back-end network emulator and can be used to simulate various Telecom settings and configurations. Such 'simulator' proxies can be used by developers for correctness and functionality testing, and debugging, before moving on to 'real' network proxies. In this case, developers do not need to change their service design, albeit just switch to the corresponding proxies for real network.

5.3.2 Billing

Telecom operators spend a significant amount of time provisioning a new service and connecting it to their billing system. In SewNet, we have exposed the Charging functionality of the Telecom operator (on top of the defined Parlay-X interfaces) using the T-Rec proxy model, and made it available through the repository. A well encapsulated proxy for billing helps the operator in populating charging rules corresponding to a proxy user. Furthermore, billing proxy can also be offered to a developer who can utilize it to charge users of her application. The charging models supported by the Billing proxy are of various types; for instance, charging could be based on a contract basis or a per usage basis.

5.3.3 Proxy Configuration

SewNet proxies include a set of configurable attributes to suit a variety of applications. For example, in SMS the developer can restrict the size of message sent by a user. For Location, the developer can configure the period of time for which the location information is cached within the proxy. Similarly, default handlers are defined for exceptions thrown by different interfaces. For instance, on encountering a *NetworkBusyException*, the default rule can be to wait for a certain amount of time before retrying. These attributes can be utilized and configured by the proxy user at any stage of the service development process to suit her requirements.

5.4 Code Generation

SewNet's Composition Studio allows the developers to output the service code in two different programming styles - Java and BPEL. In each of these cases, the service design is captured using a `processDoc` (described earlier in Section 4.3) based on the BPEL Schema. BPEL is an XML-based programming language to describe high level business processes and has many constructs to capture service design, including invocation, fault handling, correlation, and support for conditional logic.

5.4.1 Java

Service design captured in `processDoc` is converted into a Java project that can be imported inside the Eclipse programming environment. For this purpose, we first make a skeleton of an Eclipse project including creation of a package structure and the metadata files *.project* and *.classpath*. A Java class is generated for each of the Telecom and non-Telecom blocks occurring in `processDoc`, while a main class is created to capture the flow of the service. During population of the main class, constructs from the `processDoc` (invoke, switch, etc.) are transformed to the corresponding ones in Java. While for the classes generated for non-Telecom blocks, the developer is expected to embed her application logic, for the Telecom blocks, SewNet populates the classes with code for invoking the chosen proxy. For this purpose, we first import the underlying implementation jars from the repository, add the corresponding import statements in the Java class[9], add statements to instantiate the underlying proxy object, and capture the output of API chosen by the developer. Further, we parse the API invoked to import the data-types corresponding to input and output variables, and to add try-catch blocks for handling the exceptions that can be possibly thrown. In general, the code produced is well-formatted and documented to smoothen the task of ordinary Java programmers.

While the code generated acts as a template for the composite service, it needs to be examined and modified by the developer to ensure that the data flow between component services (input-output type matching, ordering of parameters, etc.) is handled properly. Further, it should be noted that since we are using a `processDoc` based on BPEL schema to capture service design, the code that can be outputted in Java is restricted to those constructs that can be represented using BPEL structure. We are currently investigating extensions to this schema that would enable a richer Java code.

5.4.2 BPEL

In this case, we assume the Telecom blocks to be realized by Web services and BPEL becomes the 'glue' to bind these Web services into a cohesive business solution. From the service design in `processDoc`, we first generate the WSDL description that provides the name and interface for the composite service and describes the port types and partner link types for stitching together the different blocks (components) in this service. The next step is the generation of a concrete BPEL structure that captures the invocation of different components in the manner as described by the `processDoc` i.e., the control-flow of the service. Further, we populate the Telecom blocks using specific details from the corresponding WSDL descriptions. For example, we introduce variables that capture the input and output of each of these blocks. Similarly, we automatically import the schemas containing definitions of the various used data-types. For the non-Telecom blocks, the developer has the option of either wrapping them as Web services or utilize extensions to BPEL supporting inline Java code. Finally, similar to the case of code generation in Java, the developer needs to edit this template BPEL workflow (for data-flow, etc.) before it is actually deployed.

6. DEMONSTRATION

We now illustrate how SewNet can be used for rapidly developing new Telecom applications. We demonstrate composition of two services. While the first one involves utilizing core Telecom functionality glued together by rich application logic, the second service is a mashup of Telecom functionality and a third party service.

6.1 Composing a 'Live' Yellow-Pages Service

Telecom operators with their defined presence and location frameworks can enable a 'dynamic, real-time' Yellow-Pages service. In essence, this service would provide matches in response to a customer request seeking 'nearby', 'available' vendors (such as tourist looking for a cab) by using Telecom network to determine the availability and location of the subscribed vendors. Business Finder [17] is an example of such a Yellow-Pages service.

[9]entries are added to the .classpath metafile also

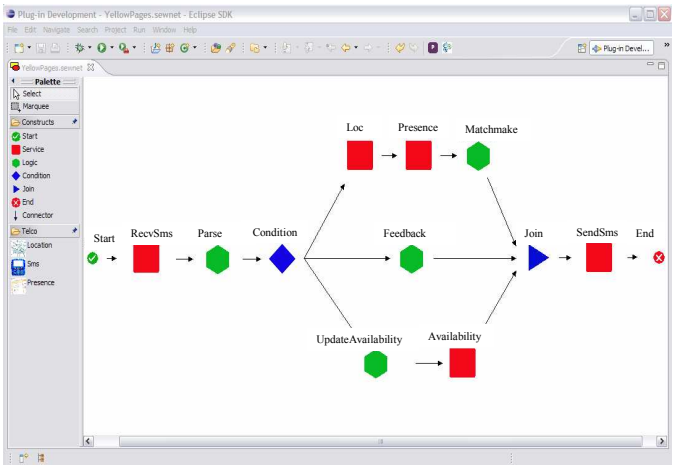

Figure 7: Yellow Pages Design

Figure 7 illustrates the design of this service on SewNet's Composition Studio. The service receives request messages via the 'receive SMS' functionality, and proceeds to parse the content of these messages. The incoming requests can be categorized into three types. The first type corresponds to a customer's request for a nearby, available vendor. For this, the service determines the current location and presence information of the subscribed vendors and formulates its response based on its internal match-making logic. The second type of request is a feedback from a customer while the third type is from a vendor asking to update her availability status. Finally, 'send SMS' is used to return the response to the requester. Once the service has been designed, the developer proceeds to look up appropriate proxies for each of the Telecom blocks. In this case, proxies are selected for SMS, Location and Presence functionalities.

From the service design, code can be generated either in Java or as a composite BPEL. Figure 8 presents a snap-shot of the code generated in Java for the designed Yellow-Pages service. As the code structure in the left panel of the figure shows, a class is created for each of the Telecom and non-Telecom blocks. While the classes for `SMS`, `Availability` and `Location` are populated with code to invoke the underlying proxy, classes for `Parse`, `Matchmake`, `Feedback` and `UpdateAvailability` correspond to the application blocks, and act as place-holders for the service developer to incorporate her application specific logic. `YellowPages.java` is the main class that coordinates the control flow of the service. This class has the code to call each proxy and application logic block using the *invoke* keyword. It also contains the transformation for constructs like switch, fork, join, etc. used by the developer in designing the service. Finally, as shown, the code also includes the Telecom constructs library (c.f. Section 5.1) and a unit test code for each Telecom proxy.

The right section of Figure 8 presents the code generated to handle invocation of the Location proxy, including the block to instantiate the proxy object, import statements to define the used data-types, and handler blocks for the exceptions thrown. In this example, we have also incorporated calls to the Billing service (for monetizing the usage of Location proxy). As outlined earlier, the code produced can be easily imported as a project and enhanced inside the Eclipse development environment.

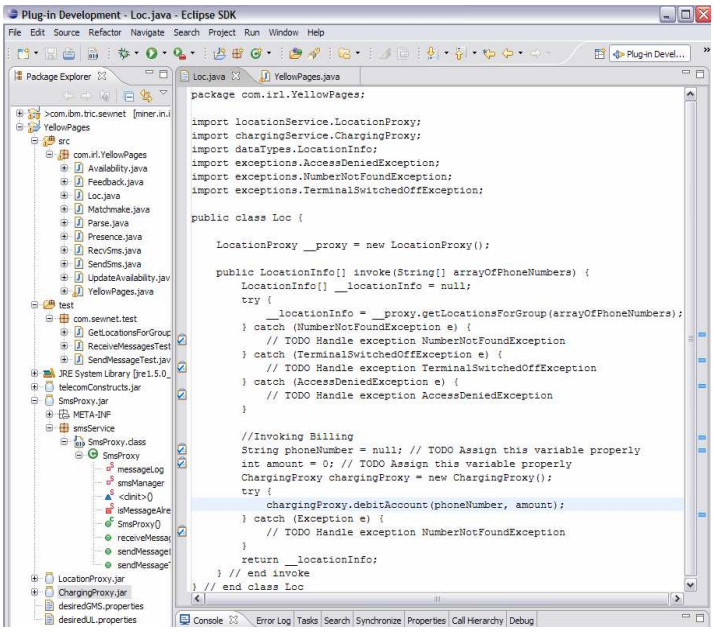

Figure 8: Code for Yellow Pages

6.2 Creating a Mashup Application

SewNet proxies can also be used to compose mashups involving Telecom functionality and third party services. An example is demonstrated in Figure 9 that combines the Location proxy in JavaScript format along with the popular Google Maps service. This simple mashup receives updates on the location information and displays it on Google Maps. Note that this currently has no application logic component. However, the same can be incorporated to develop a variety of different solutions, such as a mobile workforce management system [15][10].

We are in the process of extending an existing mashup editor, called QEDWiki [3], to enable seamless incorporation of SewNet JavaScript proxies in the mashups designed. More specifically, this editor needs to add statements to import the implementation (`locationProxy.js` file that contains `getLocation()` function), include JavaScript code to invoke the proxy (call `getLocation()` that internally subscribes `callback()` to receive location updates), define variables to capture the invocation output, etc. All this shall be automatically done through interpretation of the JavaScript proxy structure.

7. DISCUSSION AND RELATED WORK

In this section, we put SewNet under the perspective of various current and past efforts towards service composition. While some of these pertain to Web applications in general, others are more specific to the Telecom domain.

Composition using Web Services. The literature on Web service composition is extensive. Triana [20] aims to facilitate Web service composition by providing a higher level of abstraction and guiding developers in creating composed

[10]A mashup based on SewNet proxies will be demonstrated in the Conference on IP Multimedia Subsystems Architecture and Applications (IMSAA-2007).

```
<html>
<head>
.....
<script type="text/javascript" language="Javascript1.2"
src="scripts/locationProxy.js"></script>
.....
<script type="text/javascript">
var latitude;
var longitude;
function callback(){
 if(req.readyState==4)    {
        if (req.status == 200) {
                        latitude=req.responseXML.getElementById('lat');
                        longitude=req.responseXML.getElementById('long');
                }
    }
}
</script>
</head>
<body>
.....
<!-- Application specific HTML and Javascript Logic -->
<!-- Take CellNumber as Input -->
<!-- javascript locationProxy returns latitude and longitude respectively
-->
<script type="text/javascript">
getLocation(CellNumber);
</script>
<script type="text/javascript">
if (GBrowserIsCompatible()) {
        var icon = new GIcon(baseIcon);
        icon.image = "http://www.google.com/mapfiles/marker.png";
        var point = new GLatLng(latitude, longitude);
        map = new GMap2(document.getElementById(" "));
        map.setCenter(point, 13);
        map.addOverlay(new GMarker(point, icon));
        }
</script>
.....
</body>
</html>
```

Figure 9: Mashup Code Fragment

services. The paper presents a case study that investigates how this environment can be used in a Telecom setting. Synthy [14] demonstrates composition of a workforce management application enabled over a Telecom network, where core functionalities like Location are exposed as semantically annotated Web services and orchestrated using AI planning techniques. The Meteor-S [23] framework looks at a number of aspects related to composition of Web services, including capture of semantic requirements of the process and choosing components under given constraints. Zeng et al. [24] propose a method for choosing component services during Web service composition based on a generic QoS model (based on price, duration, reliability, etc.) and established linear programming techniques. As noted earlier, in a Telecom environment, we need a broader model than pure Web services to cover different protocols. In any case, SewNet's T-Rec proxy model is rich enough to apply many of these techniques. For example, each proxy can be annotated with QoS guarantees regarding its reliability, response time, etc., using which developers can estimate the QoS parameters of their composed services according to [16].

Tools for Mashup Applications. One of the most commonly used terms for Web 2.0 applications is a "mashup" - an application that combines content from more than one source into an integrated experience. Content is picked up from multiple servers (data sources) using technologies like Ajax and REST, and composed (or *rendered*) in the same user interface (UI), typically a browser. There are several tools that aid in the creation of such mashup user interfaces. Examples are QEDWiki [3], Yahoo Pipes [12], Aqualogic [2], and PrestoStudio [5]. These tools cater to a class of application developers who prefer "drag-and-drop" operations to create their own mashups. Most of these tools are not Telecom specific and can benefit by incorporating SewNet's proxies. Web21C [11] from British Telecom allows developers to integrate core Telecom functionality with other Web services into a single application, while allowing application specific logic to bind the component services. Like our work, Web21C hides the complexity of Parlay-X or SIP by exposing these Telecom functions as higher level APIs. Similarly, Connected Services Framework Sandbox [7] from Microsoft provides high level Web service interfaces via its service delivery platform that hide low level Telecom protocol specific constraints and allow creation of *Managed network mashups*. However, both these lack a structured and rich format like our T-Rec proxy model.

Telecom Standards and Platforms. The first wave of efforts along exposing core Telecom functionality primarily focused on creation of open standards and APIs. For example, Session Initiation Protocol (SIP) is an open standard being widely adopted across service providers to create Voice-over-IP services. Many Telecom operators are now moving towards implementing the IMS framework [4] to expose their core functionalities (voice service, SMS service, Call control) using SIP. JSR-289 [6] has been proposed by Sun and Ericsson to enhance existing SIP Servlet specification and support development of composed applications involving both HTTP and SIP servlets. Efforts are being undertaken to come up with platforms that support concurrent execution environments for different standard-based protocols (e.g. SOAP, SIP over HTTP) and business logic written in several languages (Java, BPEL, etc.). For example, the IBM WebSphere Application Server product suite[11] implements a converged SIP and SOAP servlet container, while enabling tailored message processing and policy enforcement for these protocols. Alcatel is also exploring the option of SOA/REST APIs on top of its Telecom Application Server [1]. From SewNet's perspective, the proxy model can encapsulate different standards, while the implementation module can be based on available platforms.

Telecom Device Functionality. A generic model of Telecom services is described in [18], where apart from core Telecom network functionality and third party offerings, services also make use of device functionality and information such as calendar, user profile and location (e.g. cell site information). Currently, there are a plethora of end-user devices in a Telecom environment, ranging from basic (offering only the capabilities of messaging and voice) to sophisticated, state-of-the-art devices like iPhone[12]. There are efforts to promote an open mobile phone software stack that abstracts hardware differences and offer a uniform set of APIs for accessing many of the functions of a mobile phone (calendaring, date, etc.). JSR 248 specification, contributed significantly by Sun Microsystems, is a step towards that direction. To compose richer Telecom applications within SewNet, we can wrap the device functionality using the proxy model and make them available to a developer during the composition process. In this case, such 'device-side' proxies will not only ease the access to device functionality, but also shield the developer from the heterogeneity of the underlying device itself. We wish to investigate device proxies based on JSR 248 in the near future.

Web 2.0 and SewNet. The phrase Web 2.0 refers to a perceived business revolution in the Web community caused by the movement to Internet as a platform. In essence, Web 2.0 changes the way in which businesses interact with its customers, by advocating the following core principles: a) provide power to the developers by offering rich, intuitive, and interactive interfaces for services based on Ajax or similar frameworks; b) create an architecture of participation that encourages consumers to add value to a service as they

[11] www.ibm.com/software/webservers/appserv/was/
[12] http://www.apple.com/iphone/

use it; and c) harness collective intelligence by facilitating collaboration and sharing among users through communities and social networks. We argue that SewNet, based on T-Rec proxies, incorporates these core Web 2.0 principles.

With several providers and community developers contributing towards development of services and its associated artifacts, it is important to help a developer with effective tools for inferencing and recommendation. While there has been a lot of work in the domain of inferencing and collaborative filtering [13, 21], SewNet opens up a new environment for applying current work as well as identifying interesting problems, for instance, those related to selection and recommendation of utility snippets for a proxy.

8. CONCLUSIONS AND FUTURE WORK

With Telecom market reaching saturation, Telecom operators can take advantage of their core functionalities like Location, Call Control, etc. These can be exposed as services and used by developers to compose rich, innovative applications. We believe that although operators are making efforts in this direction, a number of challenges still need to be addressed - existence of multiple protocols, different categories of developers, and the need to coordinate the usage of these functionalities. In this paper, we presented SewNet, a framework for creating applications exploiting Telecom functionality exposed over a converged IP network. We first provided an abstraction model, called the T-Rec 'Proxy' model, for encapsulating invocation, coordination and enrichment of the Telecom functionalities, and thereafter rendered a service creation architecture based on this model. We also demonstrated the effectiveness of SewNet for different categories of developers with the help of two use-case scenarios.

In the future, we wish to integrate SewNet with other development studios, including various mash-up environments and UML-based model driven composition tools. We also want to create proxies for some popular third party services available over the Web. Finally, we plan to encapsulate device capability under our proxy model and include the same in our service composition framework.

9. ACKNOWLEDGMENTS

We would like to acknowledge Himanshu Agrawal for developing SewNet's composition studio, and our colleagues in IBM Haifa Research Lab for helping us refine our ideas.

10. REFERENCES

[1] Alcatel Open Service Delivery Solution. http://www.eurescom.de/Parlay-OSA-products/Alcatel/Alcatel_open_service_delivery_solution.pdf.

[2] BEA AquaLogic Family of Tools. http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/aqualogic/.

[3] IBM QEDWiki. http://services.alphaworks.ibm.com/qedwiki/.

[4] IP Multimedia Subsystem (IMS) Architecture. http://www.dataconnection.com/sbc/imsarch.htm.

[5] JackBe PrestoStudio. http://www.jackbe.com/Papers/JackBe-StudioOverview.pdf.

[6] JSR 289. http://jcp.org/en/jsr/detail?id=289.

[7] Microsoft Connected Services Framework Sandbox. http://www.microsoft.com/serviceproviders/solutions/connectedservicesframework.mspx.

[8] Open Service Access (OSA); Parlay X Web Services; Part 1: Common. 3GPP TS 29.199-01.

[9] Parlay API Specification. http://www.parlay.org.

[10] Telco Web 2.0 Mashups: A New Blueprint for Service Creation. http://www.networkmashups.com/docs/ssi_0507.pdf.

[11] Web 21c sdk. http://web21c.bt.com/.

[12] Yahoo Pipes. http://pipes.yahoo.com/pipes/.

[13] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin. Incorporating Contextual Information in Recommender Systems using a Multidimensional Approach. In *proceedings of ACM Transactions on Information Systems*, volume 23, pages 103–145, 2005.

[14] V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava. A Service Creation Environment based on End to End Composition of Web Services. In *Proceedings of the 14th International World Wide Conference*, May 2005.

[15] N. Banerjee, K. Dasgupta, and S. Mukherjea. Providing Middleware Support for the Control and Co-ordination of Telecom Mashups. In *proceedings of MNCNA Workshop, Middleware Conference*, Nov' 07.

[16] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics*, 1:281–308, 2004.

[17] D. Chakraborty, K. Dasgupta, S. Mittal, A. Misra, A. Gupta, E. Newmark, and C. L. Oberle. BusinessFinder: Harnessing Presence to enable Live Yellow Pages for Small, Medium and Micro Mobile Businesses. In *proceedings of IEEE Communications Magazine, Issue on "New Directions in Networking Technologies in Emerging Economies"*, January 2007.

[18] D. Chakraborty, S. Goyal, S. Mittal, and S. Mukherjea. On the Changing Face of Service Composition in Telecom. In *proceedings of MNCNA Workshop, Middleware Conference*, November 2007.

[19] J. Rosenberg, H. Schulzrinne et al. SIP: Session Initiation Protocol. http://www.rfc-editor.org/rfc/rfc3261.txt, 2002.

[20] S. Majithia, M. Shields, I. Taylor, and I. Wang. Triana: a Graphical Web Service Composition and Execution Toolkit. In *proceedings of IEEE International Conference on Web Services*, 2004.

[21] J. Martineau, A. Java, P. Kolari, T. W. Finin, A. Joshi, and J. Mayfield. BlogVox: Learning Sentiment Classifiers. In *proceedings of AAAI Conference*, 2007.

[22] S. Staab et al. Web services: Been there, done that? *IEEE Intelligent Systems*, pages 72–85, Jan-Feb 2003.

[23] K. Sivashanmugam, J. Miller, A. Sheth, and K. Verma. Framework for Semantic Web Process Composition. In *Special Issue of the International Journal of Electronic Commerce (IJEC)*, 2004.

[24] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality Driven Web Services Composition. In *proceedings of World Wide Web Conference*, 2003.