WS-AMUSE - Web Service Architecture for Multimedia Services

Andreas Scholz, Christian Buckl,
Alfons Kemper, Alois Knoll
Institute of Informatics
Technische Universität München
Boltzmannstr. 3, D-85748 Garching, Germany
{scholza,buckl,kemper,knoll}@in.tum.de

Jörg Heuer, Martin Winter
Corporate Technology, Multimedia and Network Communication
Siemens AG
D-81730 München, München, Germany
{joerg.heuer,martin.winter}@siemens.com

ABSTRACT

Recently, a move from traditional, network specific multimedia services to IP-based solutions could be observed. Although many of these applications have similar requirements and address the same issues, individual solutions based on specialized protocols are commonly used. This specialization prohibits the extraction and reuse of common services and hinders the interoperability between services and the integration with external components.

A promising approach to overcome these disadvantages is the adoption of the service-oriented paradigm in communication protocols and a modularization into cooperating services. In this paper, we present a generic framework for multimedia applications consisting of a set of reusable Web service components, a modeling language based on finite state automata and a compiler. The results of a BPEL based prototypical implementation of a Voice-over-IP application show that the service oriented approach and the automaton based modeling language can satisfy the above mentioned criteria and ease application development through a higher level of abstraction. On the other hand our benchmarks indicate that current Web service technologies can lead to an insufficient performance, depending on the application scenario. Possible solutions to circumvent these deficiencies are presented at the end of the paper.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Design,Languages,Performance

ICSE'08, May 10–18, 2008, Leipzig, Germany.

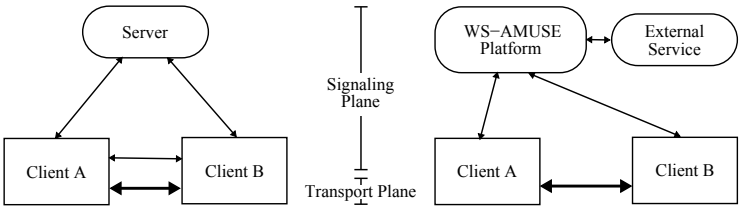
1. INTRODUCTION

In the last years, the available network bandwidth for both stationary and mobile users dramatically increased. In conjunction with reduced prices and the availability of flat-rates, a shift in the way IP based networks are used is observable. New services such as Video-on-Demand (VoD) are emerging or even replacing traditional services. A prominent example is the widespread use of Voice over IP (VoIP) applications, like SIP[21] or SKYPE[22], which replace traditional phone networks. As a consequence, telecommunication providers are more and more providing mere access to the Internet, but are excluded from the more profitable area of providing services over the Internet. Most likely, this trend will extend to mobile devices in the near future, due to the increasing availability of WLAN hotspots and built-in network cards and the availability of network technologies like UMTS.

The network providers reacted to this situation by offering services on top of network access. One such example is Triple-Play, a bundled offer of network access combined with VoD and VoIP solutions, which is offered by major telecommunication providers. A crucial observation is that these kinds of services share a lot of common functionality, e.g., for finding other users, for establishing data connections, for billing, etc. To reduce development costs and facilitate a short time-to-market, a high degree of reusability between applications is desirable. Nowadays, a typical value chain involves multiple specialized companies, e.g., for billing, authentication, delivery, etc. Besides reusability, high interoperability therefore plays an important role.

The current state-of-the-art technology for the interaction with these external services are Web services. Interestingly, the stack of Web service technologies already comprises all the needed components to not only build the interaction layer with external services, but also to build the application itself. The contribution of this paper is twofold: Firstly we present the requirements and basic building blocks of a service[1] oriented platform that allows to build Internet based applications with a high degree of reusability and extensibility and provides seamless integration with external components. The platform is intended to replace the variety of domain specific protocols used in nowadays multimedia ap-

[1]The term *service* is used differently by the telecommunication community and the service oriented computing community[16]. In this paper, we will always refer to the service oriented notion of services, i.e., a service is an encapsulated block of functionality that is directly executable and can be used in complex applications.



(a) Traditional Architecture (b) WS−AMUSE Architecture

Figure 1: Layered Architecture of Multimedia Services

plications. The modeling of services is based on interacting automatons, an abstraction that resembles the state based nature of multimedia protocols and therefore offers a fast learning curve. To simplify the implementation, executable code can be created directly from the model. Secondly, we report our experience concerning the above mentioned goals and the achievable performance for a prototypical implementation of the platform. Our primary focus are multimedia applications, but the presented concepts can also be transferred to other Internet applications.

The rest of the paper is organized as follows. Section 2 contains an overview of the structure and functionality of existing multimedia protocols. In Section 3, we refer to related work and present WS-AMUSE, our Web services architecture for multimedia services. We describe a set of reusable core Web services, a modeling language based on state automata to model the functionality and interaction of the components and a compiler for the translation of the automata into executable Web services. A prototypical implementation of a VoIP application based on this platform is presented in section 4. We report our experiences concerning the performance of the used Web service technologies in Section 5 and conclude the paper in Section 6.

2. MULTIMEDIA APPLICATIONS

A typical multimedia application can be separated into two logical planes, as illustrated in Figure 1(a). The *Signaling Plane* is used to negotiate service parameters and to control the state and the state transitions of the clients. Typically, the signaling plane is composed of one or more servers that perform user lookup, exchange presence information, establish the underlying transport connection, etc. The *Transport Plane*, on the other hand, transmits the raw payload between the individual clients of the service. The transmission is almost always performed directly between the corresponding partners, except for firewall traversal or codec translation, which may require an intermediate node. Nowadays, the Real-Time Transport Protocol (RTP)[18] is used at the transport plane in most multimedia applications. At the signaling plane domain-specific protocols like the Session Initiation Protocol (SIP)[21] and the Real Time Streaming Protocol (RTSP)[19] realize the application logic.

A crucial observation is that the two different planes have different requirements concerning the performance and reliability of data exchange. Normally, messages from the signaling plane have to be transferred reliably, i.e., the loss of messages or the duplication of messages is not acceptable. On the other hand, these messages do not have critical timing constraints, as long as the response time stays below a certain threshold. E.g., during call establishment, a delay below one second may not even be noticed by users, because the called person will need longer to pick up the call anyway. Due to the low frequency and small size of the exchanged messages, the bandwidth requirements for the signaling plane are relatively low. Quite the opposite holds for the transport plane, because a constant bandwidth with low jitter is decisive for the quality of the connection.

Note that there is a clear separation between the two planes. The signaling plane will typically only try to exchange the current IP-address with the communication partner and try to achieve a consensus concerning session parameters like used codecs, bandwidth limitations, etc. The actual data transport is done in a black-box way by the transport plane, the signaling plane only supplies the basic parameters like communication endpoints, required Quality-of-Service, etc. The only two contact points between the planes therefore are configuration options sent by the signaling plane during setup or reconfiguration or error notifications sent by the transport plane, if the given requirements cannot be fulfilled. In this work, we will concentrate on the design and implementation of the service logic, thus the signaling plane, and analyze the requirements and challenges it poses to an application developer.

Although tailored for specific applications, most of the signaling plane protocols have to solve common tasks, e.g., finding other users, storing user sessions, establishing connections between users, etc. Ideally, the developer of a new application should be able to obtain all this basic functionality by combining existing services, which may in fact be offered by specialized providers, and be free to focus on the implementation of the core functionality of the new application. To support this kind of sharing between multiple applications a modular design, which decomposes the protocols in reusable services, is necessary. Another aspect that encourages modularity is the increasing convergence between the various fields of multimedia applications. A typical example are instant messaging services, which started with simple text based chat and messaging systems. Over the time, additional functionality like document sharing, conferencing features, speech conversation, etc were added. Having a modular design for individual applications dramatically eases this kind of evolution, because extensions are clearly separated from basic building blocks and because the individual modules offer starting points for future extensions.

The signaling plane protocols themselves reveal a very similar structure and purpose: The definition of possible states for both, client and server, and the transitions between these states via the exchange of messages. The RTSP protocol, which is used in Video-on-Demand solutions, is a good example for this kind of protocol. It basically defines a set of messages that allow the user to create a RTSP stream, start or pause playback and finally close the stream. A main task of a RTSP server therefore is to store the state of all

sessions, i.e., all created streams, and update these states upon the arrival of the corresponding messages. At first glance, the state management for a phone call established via the SIP protocol is rather simple, as phone calls cannot be paused for example. However, the message exchange in the SIP protocol requires a lot of state management. The invitation of another user for example is based on a three-way handshake, which may involve multiple intermediate servers, and therefore requires every participating party to keep track of timeouts and to store which messages were sent and which replies are still missing. Considering a more advanced telephone solution, the session management for the SIP protocol can also become quite complex. Supporting functionality like holding calls and switching between concurrent calls, requires to coordinate the state transitions between multiple telephone sessions. An important requirement for a modern multimedia platform therefore is to provide a generic modeling and execution platform for this kind of state focused protocols.

The paper will demonstrate that a state automaton is a suited model to specify the behavior of a single component. To design a whole application, interacting automatons have to be used. In this case, a state transition in one automaton can be triggered by incoming external events or state transitions in other, interacting, automatons.

The presented signaling plane protocols alone are not sufficient for building a complete multimedia application. They require additional functionality, e.g., for billing, user management, searching, advertising, etc. This functionality is nowadays often provided by external, specialized providers through Web service based interfaces. To allow an easy integration with these services, the platform has to offer an easy way to access and incorporate external Web services into internal control flows.

Summing up, we identified the following three components, a flexible platform for multimedia applications should comprise:

- A modular design to allow reuse of components between multiple applications
- A toolkit to support the development of state-centric applications
- A Web service based interface for communication with services outside the platform

3. MIDDLEWARE FOR MULTIMEDIA COMMUNICATION APPLICATIONS

As mentioned in the previous chapter, modern middleware platforms for communication oriented multimedia applications have to accommodate two major requirements: First, they have to provide a Web service based interface for the interaction with external services. Second, they have to reduce the development overhead and the required time-to-market for new applications as much as possible to allow the rapid development of new applications and services. There is some previous work dealing with the integration of SIP with external services and the unification of the various signaling plane protocols for call control. However, these approaches focus on the development of VoIP like applications and only provide limited functionality for the reuse of components.

3.1 Related Work

A possible approach for providing an increased extensibility and connectivity for the SIP protocol is WSIP [13], an intermediate Web service layer that hides the details of SIP communication from the client application and provides a Web service interface for the SIP protocol, which allows an easy integration with external services. On the other hand, the actual communication on the signaling plane is still based on the SIP standard which necessitates a conversion between SOAP and SIP messages and hinders the reuse of components in other multimedia applications. Additionally, WSIP requires a Web service host on every client device. This imposes a serious overhead, especially for resource constrained devices like PDAs.

The Venice[7] project proposes a Web services based framework for VoIP applications. By using a service oriented architecture, the authors aim at easing the integration of supplementary services, the compatibility between different signaling layer protocols for call control and the installation of software updates on client devices. However, the Venice project is specifically tailored to a VoIP scenario, i.e., the authors do not address reusability between different multimedia applications nor provide a generic platform for the development of these.

Based on a distributed messaging middleware, the Global Multimedia Collaboration System (Global-MMCS)[24][26] provides a framework for an audio/video collaboration system, which bridges the gaps between nowadays multimedia applications by providing a common signaling protocol with gateways to existing protocols like SIP or H.323[6]. However, the authors do not address other multimedia applications and the reuse of components between these.

The Parlay X[14] project provides a Web services based interface for telephony systems. The aim is to provide an easy to use interface for functionality like call control, messaging, etc. This interface abstracts from the details of underlying implementation and allows a broader range of users to develop applications that incorporate telecommunication features. The actual implementation of the features is based on well known standards like SIP etc. In contrast to this, we aim at pushing the Web service approach down to this implementation layer, what removes the necessity for intermediate gateway layers and improves the interoperability between different application protocols.

3.2 WS-AMUSE Architecture

The need for reliable communication and high reusability at the signaling plane makes Web service technologies a promising foundation for building a service platform, as these already offer the needed functionality in a standardized way. Web services possess an inherent overhead due to the XML based nature of communication. But because of the relaxed performance requirements for the signaling plane, the benefits of a Web service based approach promise to outweigh possible losses in performance. The decomposition into Web services promises great advantages concerning the scalability of the system, as Web services technologies were specifically built to support a distributed execution out of the box. Additionally, Web services allow a seamless integration with external services, which are more and more often offered through Web service based interfaces. Using one single unified communication technology with both, components inside and services outside the platform, avoids

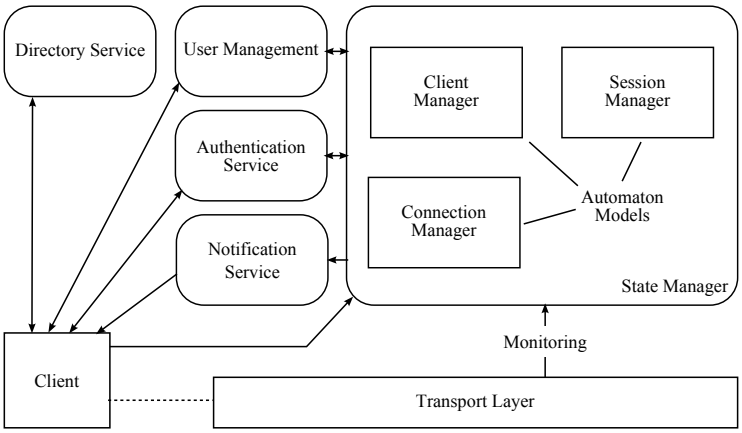


Figure 2: Components of the WS-AMUSE Platform

unnecessary breaks in communication and reduces maintenance and development costs, as there are fewer sources for errors and less technologies to master.

In this section, we will present the overall architecture and outline the most important components of WS-AMUSE, our Web service based platform designed to replace the functionality of the signaling plane protocols of nowadays multimedia applications. The actual components, i.e., Web services, may vary from application to application, but there is a common set of components that is almost always suitable.

Figure 2 gives an overview of the basic building blocks our platform currently offers. The *State Manager* is a general purpose state engine that stores and alters the state of the multimedia application and controls the interaction between the different components of the platform. In a VoIP scenario it will manage the state of individual calls and allow the client to handle multiple calls at the same time by putting calls on hold etc. With the help of the *Notification Service*, the State Manager and other components of the platform can send notifications back to the client, e.g. if an incoming call is detected. A detailed description of the State Manager and the Notification service is presented in the following sections. The *User Management* allows an application to store application specific data for every single client, e.g. the address of an in-network answering machine in a VoIP scenario. The *Authentication Service* issues security tokens that allow authenticated users to access the services on the platform. Additionally, these tokens can be used to trace the service usage of every user, and therefore build the foundation for a fine-grained billing system. The *Directory Service* provides some kind of "yellow-pages" functionality, i.e., it lists all available services on the platform and their technical characteristics, such as server addresses, etc. Note that in contrast to a UDDI-Registry, which stores information about individual Web services, the Directory Service stores information about application level services, e.g., a VoD service, a VoIP service, a chat application and so on. The *Transport Layer* itself is not directly part of the platform, as the actual data transmission is conducted by common transport protocols. However the signaling layer can use available monitoring information to deal with insufficient Quality of Service or communication failures, for example by downgrading the used audio/video codecs.

Note that the presented components only represent a basic subset of the components a productive implementation of the platform will possess. The advantages of a truly service oriented architecture will increase with every additional service in the platform. This is due to the fact that all applications developed by combining individual components of the platform can be treated as services themselves and therefore can subsequently be used to build even more sophisticated solutions. A news station, for example, could reuse an existing VoD service to distribute news broadcasts.

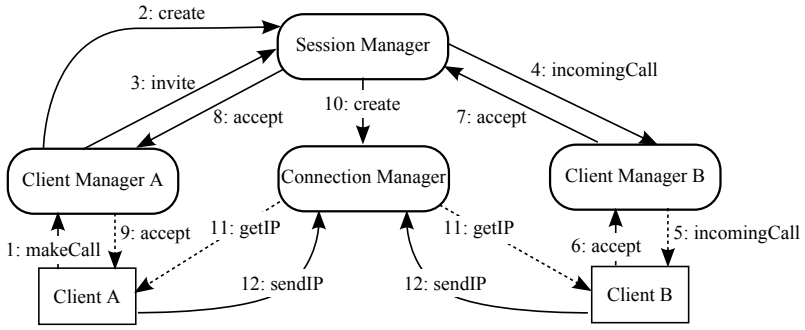
In the following, we will primarily focus on the Notification Service and the State Manager, because these two components orchestrate the interaction between the individual components of the platform and the connected users and therefore constitute the core of our architecture.

3.3 Client Notification with Stateless WS Calls

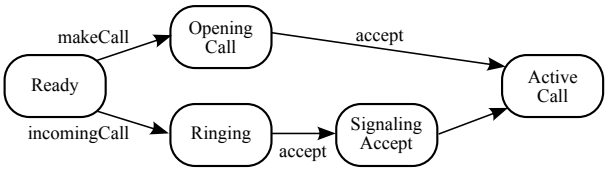
The Notification Service is necessary due to the stateless nature of HTTP based Web service interactions. Every client can decide to use non-persistent HTTP connections to submit calls to Web services on the platform, so there is no feedback channel that allows a Web service to actively send data back to its clients. One possible solution would be the installation of a Web service engine on every client. In this scenario, data can be submitted to clients by simply invoking a Web service on the corresponding client's machine. We do not think that this approach is feasible for resource constrained devices, e.g. mobile phones. Therefore, we developed a pull based mechanism. Clients periodically request new messages from the Notification Service via a Web service call. This call will block, until a new notification arrives, or a timeout occurs. In case of a new notification, the Web service call will immediately return and therefore ensure a fast delivery of notifications in the platform. In case of a timeout, the client must immediately re-issue the request. This mechanism allows to detect disconnected clients, which can be automatically removed from the Notification Service if no request was received for a tunable period of time.

3.4 Generic State Control for Web Service Architectures

We are using state automata to model the application functionality. Finite state automata offer a concise and intuitive way to model state-based applications from a developers point of view, especially because most application engineers have a background in control theory rather than in programming. State automatons are a familiar modeling



(a) Interaction between Managers



(b) Simplified Client Manager State Automaton

Figure 3: VoIP Application

paradigm for these engineers. WS-AMUSE uses SCXML[20], an XML based standard for state chart notations based on Harel State Tables, with some minor extension for providing better support for Web service architectures. In SCXML, transitions between states can be executed spontaneously, be time-triggered or be triggered by events stemming from user actions or interacting automata. In either case, an optional guard condition can be added, which ensures that certain requirements are met. The states themselves may contain optional *onEntry* and *onExit* blocks, which are executed upon entry or exit of a state. Within these blocks, the actual application logic can be implemented by using typical control flow statements like branches, loops, invocations of other Web services, etc.

Note that the automaton models are self contained, i.e., code generated from these models is directly executable without further changes by the developer. Using an additional abstraction layer, like the automaton model, is a well known technique from the area of domain-specific languages (in fact one can see the automaton model as a domain specific language for state control in multimedia applications). As shown in [23], the elevated abstraction layer can lead to development times that are 3-10 times faster than the development times of standard processes, because it allows the developer to focus on the design of a solution and hides the complexity and details of the implementation of the models.

To support this kind of model based development, the WS-AMUSE platform offers a generic state engine, the State Manager already introduced in Figure 2. To distinguish between different instances of the State Manager, we will use the name of the automaton model, i.e., the Session Manager refers to a State Manager instance executing the automaton for session management.

Summing up, the WS-AMUSE platform reduces development effort by combining two approaches: The use of service oriented principles to increase reusability, in this case a Web service based architecture, and a state based modeling language that hides communication details from the developer and allows to focus on the actual application logic.

4. IMPLEMENTATION

In this section, a prototypical implementation of a VoIP application based on our platform is described, consisting of the State Manager, a simple version of the User Management and the Notification Service.

4.1 Automaton Model for VoIP Applications

Figure 3(a) shows the three inter-operating automata used to control the VoIP application: A *Client Manager* that controls the state of a single participant, a *Session Manager* that manages a call between two participants, and a *Connection Manager* that serves as an interface to the underlying transport layer.

Figure 3(a) shows the message flow for establishing a call between user A and B. Figure 3(b) shows the simplified state automaton for the Client Managers used in this application. To provide a concise example, we only present the Client Manager, as this is the most interesting of all three automata, and do not show the edges for the transitions that allow to abort calls or handle errors. Initially, A invokes the *makeCall*(1) method on its Client Manager. This causes Client Manager A to change its state from *Ready* to *Opening Call*. In the onEntry block of the *Opening Call* state, the Client Manager A creates(2) a new Session Manager instance and subsequently submits an invitation(3) for user B. The Session Manager notifies(4) the Client Manager B of the incoming call from A, which in turn forwards(5) the notification to Client B. The forward is done via the Notification Service (indicated by the dotted line), as there is no possibility for Client Manager B to directly send messages to Client B. Client B decides to take the call and submits an accept(6) message, which is propagated(7,8) back to Client Manager A. At Client Manager A, this will cause a state change from *Opening Call* to *Active Call*. In the onEntry block of *Active Call*, the Notification Service is used to inform(9) Client A about the successful call establishment. Concurrently, the Session Manager creates(10) a new Connection Manager instance, which requests(11) the current IP-addresses of A and B via the Notification Service. By calling the *sendIP*(12)

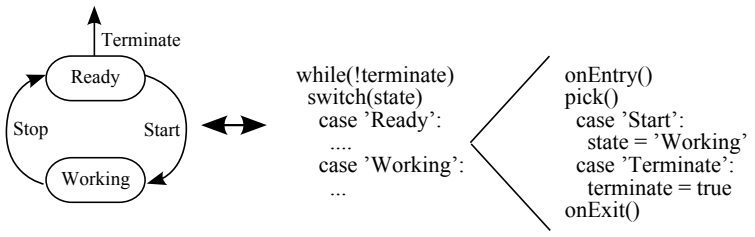


Figure 4: Transformation from Automaton to BPEL

method, both clients submit their current IP-address and in turn receive the IP-address of their communication partner (*sendIP* will block until the IP-address of the partner is received).

4.2 Implementation of the Platform

There are several possibilities to implement the state engine inside the State Manager. A straightforward approach is to encapsulate a state engine implementation inside a Web service. For our prototypical implementation, we chose a different solution based on the Business Process Execution Language (BPEL)[4]. BPEL is a platform independent state-of-the-art service composition language and widely supported by various Web service implementations. BPEL offers several advantages compared to the straightforward implementation. First of all, most BPEL engines already provide essential functionality that would have to be reimplemented otherwise: persistence of processes, addressing of specific process instances through IDs, support for additional WS-standards like WS-Addressing, WS-Security, etc. Additionally, most engines possess sophisticated auditing and logging facilities that can ease debugging during development time. Furthermore, using an execution engine based on the standardized BPEL syntax offers the possibility to benefit from future improvements in the engine implementations and to always use the most suitable platform, due to vendor independence.

Note that the BPEL language itself only provides basic programming constructs known from imperative programming and therefore does not provide an intuitive way to model state automatons. Due to this reason, we use SCXML to specify the automatons. By transforming these automatons to BPEL, we are able to use existing BPEL engines for the execution of the automaton models.

The transformation consists of two tasks: It has to create code comprising the application logic in BPEL syntax, and has to map the automaton model to a Web service based platform, i.e., convert events in the automaton model to Web service calls in the platform. BPEL itself already offers all required building blocks of a state engine: A while loop, a switch statement and a pick statement that blocks the process until one of a set of events occurs. Both, the SCXML modeling language and the BPEL execution language are based on XML, so this transformation can be realized with the standardized XSL Transformations[28] (XSLT) language. The basic structure of a transformed automaton is illustrated in Figure 4.

The BPEL workflow for a transformed automaton consists of an outer while loop that is executed until a terminating state is reached. Inside this loop, a variable is used to store the current state the automaton is in. This variable serves as input for a switch statement, which selects a code block for the current state. Pseudo code for this loop is shown in the center of Figure 4. Every state of the automaton is transformed to a block consisting of three elements: an *onEntry*-block, a pick-statement and an *onExit*-block. The *pick* statement contains another switch statement to select the appropriate action for every possible transition. The corresponding pseudo-code is shown on the right side of the Figure 4. During the transformation, all events from the automaton model are transformed into WS-calls in the BPEL engine, i.e., sending an event to the State Manager is done by calling the respective WS-method in the BPEL engine.

To exemplify the sequence of actions during a state transition in the BPEL process, let us assume the automaton is currently in the state *Ready*, i.e., the process is currently blocked at the *pick*-statement of the *Ready* state. The user now submits a WS-call to the *Start* method. The *switch* statement inside the pick will use the branch for *Start* and therefore set the current state to *Working*. After that, the *onExit* block is executed. The outer switch statement will now branch to the code for the *Working* state, execute the corresponding *onEntry* method and finally block at the *pick* statement, until a WS call to the *Stop* method is received (*Stop* is the only valid transition from *Working*).

For our prototype, we implemented two different clients: A Java based client for a traditional PC and a C++ based client for Windows Mobile 2005 PDAs. The PC client uses Axis2[2] to access the Web services on the control plane and the Java Media Framework API[10] (JMF) to send and receive data on the transport plane. The PDA based solution uses the gSOAP[5] Web service stack and the RTP stack from the PJSIP[17] library. Both clients use the Real-Time Transport Protocol for data transmission, as this is the defacto standard for the transmission of streaming multimedia data. The server side is based on the open source BPEL Engine ActiveBPEL[1].

5. EVALUATION

Based on the prototypical implementation, we evaluated the WS-AMUSE approach with regard to the achievable performance, reusability and extensibility.

5.1 Performance Considerations

First of all, we measured the raw throughput of the BPEL engine for a simple BPEL service, as this is the upper bound for all applications build on top of the BPEL engine. On our benchmark system (Apache Tomcat running on a Xeon 2.8 GHz Blade with 1GB RAM), ActiveBPEL was capable of executing approximately 250 invocations/s of a simple HelloWorld-style BPEL process using in-memory persistence. For comparison, this throughput has the same order

of magnitude as the published throughput numbers of other BPEL engines like Cape Clear ESB[9] (5100 TPM = 85 invocations/s) or Intalio BPMS[8] (17M non-persistent processes in 24h = 197 invocations/s).

To measure the achievable number of call establishments per second for the VoIP scenario, we designed a benchmark where all BPEL processes and the Notification service were located on a single blade. Another blade is used to simulate pairs of clients that repeatedly call each other, accept the call and immediately hang up afterwards. Using this benchmark, our system was capable to establish and tear down 2.3 phone calls/s. A single call establishment between two partners incorporates multiple BPEL calls for the interaction between the three state automata: For call establishment 12 BPEL calls and 8 WS calls, including communication with the notification service and exchange of IP-addresses are necessary. A call hangup requires 5 BPEL and 2 WS calls, yielding a total of 17 BPEL and 10 WS calls per phone call.

Taking into account the numbers form the HelloWorld application, this shows a considerable overhead for the execution of the BPEL processes. We were able to identify two major causes for this overhead: In contrast to the HelloWorld application, the BPEL engine also has to perform client-side parsing of SOAP messages, due to calls to other BPEL processes. Second, the BPEL processes for the VoIP scenario are much more complex and contain a lot of XPath statements for variable assignment and state management.

A major benefit of the Web service based solution is the strict encapsulation of functionality in cooperating services. The WS-AMUSE approach promises to provide scalability through the distribution of different managers or BPEL instances over multiple hosts, without additional development overhead. By simply allocating every manager service on a different host, we were able to increase the throughput up to 5.9 calls/s. Because the complexity of the state automata, and therefore the load on the corresponding machine, is not uniform, the throughput can be further improved by allocating a mix of manager instances at every host.

The throughput of 2.3 phone calls/s for the BPEL based solution is not sufficient for building a larger scale phone or conferencing solution with hundreds or thousands of simultaneous telephone calls, even taking into account a possible replication of services and more powerful servers. On the other hand, the performance achievable with the used tools may already be sufficient for other application scenarios with longer session durations, like a VoD solution for example. A VoD application is much less complex than the VoIP application and requires less coordination between the different managers because there is only a single client involved in every session. Additionally VoD sessions are changed rarely in comparison to phone calls, simply because a typical video session will last longer than a typical phone call.

To circumvent the current performance deficiencies, there are two principle approaches: the development of a state engine not based on BPEL, or the tuning of existing BPEL engines. Due to the strict separation between the modeling language and the execution language, the former can be easily achieved, e.g., by building a transformation of the automaton model to a Java based Web service instead of a BPEL workflow. In fact, the target of the transformation could be any other programming language or middleware platform too, as long as it supports some kind of message exchange between different automaton instances and Web service calls to remote services. However the drawback of these solutions is that one looses the high level of abstraction the BPEL based solution offers. In these cases, special care has to be taken that the integration with other business processes and Web services, which BPEL offers out of the box, is still supported by the automaton implementation.

Despite these performance issues, we think the BPEL approach can be a feasible solution even for large scale installations. The currently available engines are built to support the coordination of business services and therefore induce a lot of logging and auditing functionality. Although handy for development, this functionality is not needed in a productive environment because the individual Web service invocations are only meaningful in a certain context and storing them provides no benefits. E.g., the exchanged messages for a call establishment are individually of no interest, the only noteworthy result is the fact that the establishment of the call was successful or not. This is not the case for business services, as interactions between these normally have a business related background, which makes every single call important. Another area for optimization is process persistence. In contrast to business services, multimedia applications typically have short session durations, but a relatively high invocation rate. E.g., a VoD session will seldomly last more than two or three hours, as most movies have a shorter length, but it is quite likely that the user will pause the playback at least once. In the business process scenario, the execution of a whole workflow may take several days, especially if human interaction is necessary, but most of this time is spent waiting for input from external sources. We think that a BPEL engine specially tuned for these characteristics could provide a significant performance boost.

On the messaging layer, the XML processing of SOAP messages is a major performance bottleneck. Currently there are several projects aiming at improving the performance of XML message handling. One possibility is the use of binary XML representations like BiM[3] or WBXML[25] for example. We did some simple benchmarks with the kXML[12] WBXML parser, which performed five times faster than the Xerces[27] parser used by the Active BPEL engine, while still providing a self contained representation of the SOAP message. Another possibility is to exploit schema information to build specific XML parsers[15] or the fact that most SOAP invocations to a service will only differ in small parts of the message body[11].

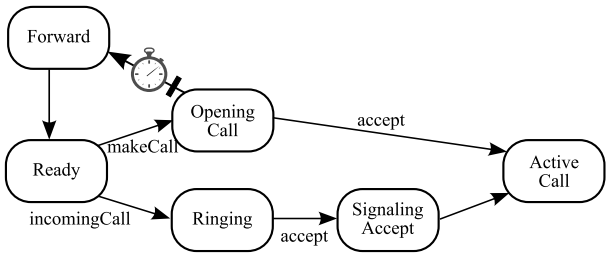
Summing up, we determined that the currently available BPEL engines are not yet capable of coordinating the signaling plane of large scale multimedia applications. But we are confident that this will be the case with some optimizations concerning persistence and auditing functionality, accompanied by optimizations for the XML processing.

5.2 Extensibility and Reusability

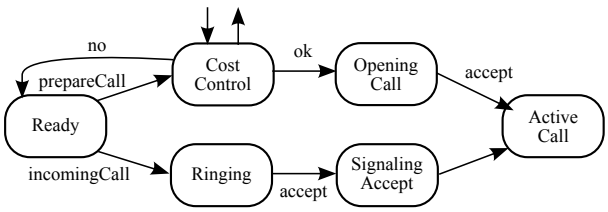
One aim of the WS-AMUSE project was to increase the extensibility and reusability of multimedia applications. The extensibility can be seen at three levels: at the code synthesis, the augmentation of existing services and the addition of new services.

Adapting Code Generation

Due to the model-based approach, it is relatively easy to perform changes that do not change the functionality, but rather the implementation of an application. This is for



(a) VoIP Client Manager with Forwarding Functionality



(b) VoIP Client Manager with Cost Control

Figure 5: Augmented Versions of the VoIP Client Manager

example very useful, when the application code must be adapted to the used platform, or new functionality should be added to many nodes simultaneously. To illustrate this possibility, we added a debug mode to the code generator.

In debug mode, the state engine will report the received event and the new state to a user defined application. We implemented a simple tool that is able to draw an automaton model and store the sequence of reported events, allowing a developer to inspect the state changes and received messages throughout the whole lifetime of a state manager. Knowing the sequence of state changes and exchanged messages eases debugging and helps in finding performance bottlenecks. Note that this adaptation was performed only by augmenting the code generator. The application model could be reused without any changes.

Augmenting Existing Services

The model-based approach also simplifies the augmentation of existing services. Due to the high abstraction level, changes of the behavior can be done very easily. As an example, we chose to add a forwarding functionality to the client manager. An incoming call is forwarded to a third party (e.g. the mobile phone or an answering machine), if the user does not respond during a specified time interval. The extended model is illustrated in Figure 5(a). The transition to the state *Forward* is triggered by time. In addition, this transition can only be triggered, if a forwarding address is available (guarded transition). The state *Forward* triggers a new event that initiates an incoming call to the third party and informs the calling client that the call was forwarded. Afterwards the transition to state *Ready* is performed immediately (spontaneous transition) and the corresponding client is ready to receive the next call.

Integrating New Services

A big benefit of the Web service based communication and the BPEL based application logic is the possibility to seamlessly integrate external Web services at every location in the control flow. For some scenarios this offers only little benefits compared to a SIP based solution. For example consider the integration of a billing service into the call establishment. In this case, a simple Web service interface for the external service is sufficient, i.e., by extending a traditional SIP based application with a Web service stack this task can be solved.

On the other hand this is not the case for all extensions that require an interaction with the end-user. Assume we want to extend the VoIP application described in Section 4 with a cost control service, i.e., the user should be able to see the costs per minute of the current phone call and be able to accept or reject the call establishment based on these costs. To incorporate this functionality, the addition of a single new state is sufficient, as shown in Figure 5(b). After the user has specified the person he wants to call (*prepareCall*), the Client Manager enters the *Cost Control* state and requests the costs of the call from a billing Web service, or some similar source. Based on the costs displayed on the client device, the user can decide to accept the price (*ok*), in which case the call establishment proceeds, or to reject the price (*no*), in which case the call is aborted and the Client Manager enters the *Ready* state again.

Adding this kind of functionality to a SIP based solution is much more complex, as the developer would have to extend the SIP protocol with new messages, build a Web service based interface to the billing service and finally convert the client messages to Web service calls and the responses back to SIP again. A major contribution to the extensibility of the WS-AMUSE platform therefore is the unified communication via SOAP, which allows to incorporate new services at any place in an existing application, without the need to convert messages to an internal protocol.

Increasing Reusability

Evaluating the reusability of components is always a difficult task. From our experience with the development of the prototypical VoIP application, we see multiple sources of reuse. First, whole blocks of functionality can be reused by different applications, e.g., the authentication functionality, the user management, etc. The second source of reusable components are fine grained services, like the Connection Manager for example. These services are no standalone components, but provide essential functionality which is needed in many applications, like the establishment and monitoring of transport layer connections in the Connection Manager for example. Both cases are well supported by the service oriented paradigm used for the development of the applications, as it encourages the encapsulation of functionality in small, self-contained services. The third possibility of reuse are the state automatons developed for individual applications. A lot of control tasks are quite similar, so parts of existing state automatons can be used to build new applications. E.g., the Session Manager of a VoIP application could be a good starting point for the development of a conferencing application, as it already possesses the basic functionality for the establishment of sessions containing two members and should be easily extendable to multi user sessions.

6. CONCLUSION

In this paper, we presented the requirements and development challenges of a next generation service platform, which

is capable of replacing traditional, network protocol oriented applications like Voice-over-IP or Video-on-Demand. The presented flexible and reusable framework fosters the rapid development of future multimedia applications. WS-AMUSE offers a Web service based platform that leverages the functionality of existing multimedia applications and additionally provides a higher flexibility, a better reusability and a seamless integration with external services. This is done by defining a set of default components that can be reused by various applications, and by using a concise automaton based modeling language that allows to focus on the development of the application logic. We developed a BPEL-based prototypical implementation of the platform for a VoIP scenario and reported our experiences concerning the reuse of components and the extension of existing applications. In both cases, the service oriented paradigm and the unified Web service based communication reduce the development costs. On the other hand, a performance evaluation done with the prototype shows that with the currently existing technologies the BPEL based approach may not be feasible for all applications, e.g., high throughput applications like large scale phone solutions. We outlined several optimization possibilities to circumvent this issue, either by tuning the underlying BPEL engine or by speeding the XML processing on the messaging layer, e.g., by using binary XML or exploiting regularities in the structure of SOAP messages.

There are several directions for further research. First of all the management of user data in service oriented architectures has to be studied further. There already are approaches concerning federated identity management or identity management for communities heading in a similar direction, but these do not take into account the composite nature of service oriented architectures. E.g., it might be the case that a service is only allowed to indirectly access banking details of a user, i.e., it may be allowed to pass this data on to a banking service, but not to read the contents. To allow the use of a BPEL based solution in high throughput scenarios, possible improvements for the performance of the underlying Web service technologies, as outlined Section 5.1, have to be evaluated. To provide compatibility with existing SIP or RTSP based solutions, it would be useful to implement gateways that translate between messages from the network protocols and Web service calls from the WS-AMUSE platform. Furthermore, the service oriented approach can be extended to other application scenarios as well, e.g., to online games or conferencing systems.

7. REFERENCES

[1] ActiveBPEL. http://activebpel.org/.

[2] Axis2. http://ws.apache.org/axis2/.

[3] Binary MPEG format for XML, ISO Reference Number: ISO/IEC FDIS 23001-1:2006(E), Part 1.

[4] BPEL, Business Process Execution Language. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[5] gSOAP. http://gsoap2.sourceforge.net/.

[6] ITU. Recomm. H.323 (1999), Packet-base multimedia communications systems.

[7] M. Hillenbrand, J. Götze, and P. Müller. Voice over IP - Considerations for a Next Generation Architecture. In *EUROMICRO-SEAA*, pages 386–395. IEEE Computer Society, 2005.

[8] Intalio Inc. Intalio BPMS sets the standard for BPM performance. http://www.intalio.com/news/press-release/?release=20061121-Benchmark.

[9] Intel and Cape Clear. BPEL scalability and performance testing White Paper. http://www.capeclear.com/download/portal.php.

[10] JMF, Java Media Framework. http://java.sun.com/products/java-media/jmf/.

[11] C.-C. Kanne and G. Moerkotte. Template Folding for XPath. In *Third International Workshop on XQuery Implementation, Experience and Perspectives (XIME-P 2006)*, 2006.

[12] kXML2. http://kxml.sourceforge.net/kxml2/.

[13] F. Liu, W. Chou, L. Li, and J. J. Li. WSIP - Web Service SIP Endpoint for Converged Multimedia/Multimodal Communication over IP. In *Proc. of the IEEE Int. Conf. on Web Services*, pages 690–699. IEEE Computer Society, 2004.

[14] H. Lofthouse, M. J. Yates, and R. Stretch. Parlay X Web Services. In *BT Technology Journal*, volume 22, pages 81–86, 2004.

[15] W. Löwe, M. L. Noga, and T. Gaul. Foundations of Fast Communication via XML. In *Annals of Software Engineering*, volume 13(1-4), pages 357–379, 2002.

[16] T. Margaria, B. Steffen, and M. Reitenspieß. Service-oriented design: The roots. In *Proc. of the 3rd Int. Conf. on Service Oriented Computing*, volume 3826 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2005.

[17] PJSIP. http://www.pjsip.org/.

[18] RTP, Real-Time Transport Protocol. http://www.ietf.org/rfc/rfc3550.txt.

[19] RTSP, Real Time Streaming Protocol. http://www.ietf.org/rfc/rfc2326.txt.

[20] SCXML, State Chart XML. http://www.w3.org/TR/2005/WD-scxml-20050705/.

[21] SIP, Session Initiation Protocol. http://www.ietf.org/rfc/rfc3261.txt.

[22] SKYPE. http://www.skype.com/.

[23] J.-P. Tolvanen. Making model-based code generation work. *EmbeddedSystems Europe*, pages 36–38, Aug 2004.

[24] A. Uyar, W. Wu, H. Bulut, and G. Fox. Service-Oriented Architecture for a Scalable Videoconferencing System. In *Proc. of IEEE Int. Conf. on Pervasive Services 2005*, pages 445–448. IEEE Computer Society, 2005.

[25] WAP Binary XML Content Format. http://www.w3.org/TR/wbxml/.

[26] W. Wu, G. Fox, H. Bulut, A. Uyar, and T. Huang. Service Oriented Architecture for VoIP conferencing. In *Int. Journal of Communication Systems, Special Issue on Voice over IP - Theory and Practice*, volume 19, pages 445–462. John Wiley & Sons, 2006.

[27] XERCES. http://xerces.apache.org/xerces-j/.

[28] XSLT, XSL Transformations. http://www.w3.org/TR/xslt.