A scalable service-oriented architecture for multimedia analysis, synthesis and consumption

Steffen Heinzl*

Department of Mathematics and Computer Science
University of Marburg
Hans-Meerwein-Str. 3
D-35032 Marburg, Germany
E-mail: heinzl@informatik.uni-marburg.de
*Corresponding author

Dominik Seiler

Department of Mathematics and Computer Science
University of Marburg
Hans-Meerwein-Str. 3
D-35032 Marburg, Germany
and
Information Systems Institute
University of Siegen
Hölderlinstr. 3
D-57068 Siegen, Germany
E-mail: seiler@informatik.uni-marburg.de
E-mail: d.seiler@fb5-uni-siegen.de

Ernst Juhnke, Thilo Stadelmann and Ralph Ewerth

Department of Mathematics and Computer Science
University of Marburg
Hans-Meerwein-Str. 3
D-35032 Marburg, Germany
E-mail: ejuhnke@informatik.uni-marburg.de
E-mail: stadelmann@informatik.uni-marburg.de
E-mail: ewerth@informatik.uni-marburg.de

Manfred Grauer

Information Systems Institute
University of Siegen
Hölderlinstr. 3
D-57068 Siegen, Germany
E-mail: grauer@fb5.uni-siegen.de

Bernd Freisleben

Department of Mathematics and Computer Science
University of Marburg
Hans-Meerwein-Str. 3
D-35032 Marburg, Germany
E-mail: freisleb@informatik.uni-marburg.de

Abstract: Although Service-Oriented Architectures (SOAs) were not designed for multimedia processing, they speed up the development of distributed multimedia applications by allowing the composition or reconfiguration of existing services. For example, the Business Process Execution Language for Web Services (BPEL) is a powerful tool to orchestrate, model and execute workflows. However, due to its process-oriented approach, it is not directly applicable to data-intensive applications, such as those from the multimedia domain. In this paper, a comprehensive service-oriented infrastructure for multimedia applications is presented that (a) overcomes some drawbacks of BPEL for data-intensive applications and (b) provides tools that further ease the development and use of web services for a broad scope of multimedia applications covering video content analysis, audio analysis and synthesis and multimedia consumption. The proposed service-oriented infrastructure can be easily integrated into existing business processes by using BPEL. A dynamic allocation of cloud computing resources ensures the scalability of a multimedia application. To allow efficient and flexible data transfers in BPEL workflows, an implementation of the Flexible SOAP with Attachments (Flex-SwA) architecture is used that allows data transmission in conjunction with SOAP messages. The protocol requirements of services in the case of real-time, streaming or file transfer can be described by a communication policy. Three use cases of multimedia applications are evaluated.

Keywords: multimedia; service-oriented architectures; SOAs; distributed analysis; mash-up; workflow; Business Process Execution Language; BPEL; Visual Grid Orchestrator; ViGO; video content analysis; audio resynthesis.

Reference to this paper should be made as follows: Heinzl, S., Seiler, D., Juhnke, E., Stadelmann, T., Ewerth, R., Grauer, M. and Freisleben, B. (2009) 'A scalable service-oriented architecture for multimedia analysis, synthesis and consumption', *Int. J. Web and Grid Services*, Vol. 5, No. 3, pp.219–260.

Biographical notes: Steffen Heinzl is a Research Assistant at the Department of Mathematics and Computer Science of the University of Marburg, Germany. He received his Diploma in Computer Science from the University of Applied Sciences, Fulda, Germany, in 2004. He is currently pursuing his PhD at the University of Marburg. His research interests include web services, web service policies, data transfers and usability in service-oriented environments and the combination of multimedia and the SOA paradigm.

Dominik Seiler received his Diploma in Computer Science in 2008 from the University of Marburg, Germany. He is currently working as a Research Assistant at the Information Systems Institute of the University of Siegen, Germany, and pursuing his PhD at the University of Marburg. His research interests include SOA, cluster/grid computing and distributed multimedia processing.

Ernst Juhnke is a Research Assistant at the Department of Mathematics and Computer Science of the University of Marburg, Germany. He is currently pursuing his PhD at the University of Marburg, where he received his Diploma in 2007. His research interests include SOA, workflows and cluster/grid computing.

Thilo Stadelmann received his Diploma in Computer Science from the Giessen-Friedberg University of Applied Sciences, Germany, in 2004. He is currently working towards his PhD at the University of Marburg, Germany, where he is also a Research Assistant at the Department of Mathematics and Computer Science. His current research interests include artificial intelligence, pattern recognition and machine learning, especially in the field of automatic speaker recognition.

Ralph Ewerth received his Diploma in Computer Science in 2002 and his PhD in Computer Science in 2008, both from the University of Marburg, Germany. He is currently a Research Assistant at the Department of Mathematics and Computer Science of the University of Marburg. His research interests include distributed multimedia processing and content-based multimedia retrieval.

Manfred Grauer received his Diploma in Engineering/Cybernetics in 1970 from the Moscow Institute of Technology, Moscow, Russia, and his PhD in 1975 and his Habilitation in 1979 from the Technical University of Merseburg, Germany. Since 1989, he has been the Director of the Information Systems Institute of the University of Siegen, Germany. From 1991 to 1995, he was elected as the Dean of the Faculty and served from 1997 to 2002 and again from 2006 until now as a Vice President of the University of Siegen, Germany. His research interests shifted from interactive decision systems towards the design of information systems and he is currently focusing on product life cycle management and cluster/grid computing, including multidisciplinary analysis and optimisation.

Bernd Freisleben is a Full Professor of Computer Science at the Department of Mathematics and Computer Science of the University of Marburg, Germany. He received his MSc in Computer Science from Pennsylvania State University, USA, in 1981 and his PhD and Habilitation in Computer Science from Darmstadt University of Technology, Germany, in 1985 and 1993, respectively. His research interests include distributed/parallel systems, cluster/network/grid computing and middleware for internet application development.

1 Introduction

With the continuous growth of video data, there is an increasing demand for video content analysis and indexing to effectively support video databases and retrieval systems. The typical tasks for video content analysis (such as cut, face or text detection) are computationally demanding and require distributed (high-performance) resources for parallel processing to guarantee an acceptable run-time behaviour. Since media analysis deals with computation-intensive tasks, it is reasonable to divide the analysis as a whole into several steps, such that single analysis steps can be distributed to different nodes. The paradigm of a Service-Oriented Architecture (SOA) promises that these computation-intensive tasks can be exposed as services and effectively combined to new applications, thus speeding up the development of applications. For the combination of these services (called ‘orchestration’), it is reasonable to use the Business Process Execution Language for Web Services (BPEL), the *de facto* standard for workflows in the industry, to allow companies to integrate multimedia services into their existing service portfolios. However, since all service data pass the BPEL engine, the application of BPEL to data-intensive applications from the multimedia domain is not very efficient. Furthermore, the development of web services is still difficult and time-consuming and in practice, it is almost impossible for ordinary end users. To obtain a broader user basis, it is necessary to simplify the use of web services. Also, to provide a timely execution, a distributed infrastructure for multimedia processing should be easily scalable.

In this paper, we present an SOA for multimedia applications to address these issues. The used Flexible SOAP with Attachments (Flex-SwA) framework enables the modelling of data flows in BPEL. In this way, an efficient and flexible data transfer is possible in BPEL workflows for multimedia applications. In addition, several tools are offered to ease the development of web services, namely:

- a web/grid service browser
- a mash-up editor to easily use services
- the Visual Grid Orchestrator (ViGO) to create new services from existing ones.

Scalability is achieved by the possibility of dynamically allocating resources from a computational cloud, such as the Amazon Elastic Compute Cloud (EC2) (Amazon Web Services LLC, 2009a). Finally, it is shown how a communication policy (Heinzl *et al.*, 2008a) embedded in the service’s Web Services Description Language (WSDL) document can be used to describe the protocol requirements of services to support real-time, streaming or file transfer. Three case studies covering a broad multimedia scope are presented to show how technologies and tools provide conceptual improvements as well as improvements in terms of usability, ease of development and efficiency. An audio resynthesis service, a face detector service and a video-on-demand service are analysed in detail, along with the experimental results.

The rest of the paper is organised as follows. Section 2 discusses the requirements for a multimedia SOA, introduces a layer model for it and explains the tools and technologies needed to fulfil the requirements. Section 3 presents the services that the multimedia SOA currently offers and describes the services that have been selected as use cases, as well as their implementation. Section 4 presents a quantitative evaluation of the use cases, whereas Section 5 presents a qualitative evaluation providing deeper insights into the feasibility of the provided tools. Section 6 gives an overview of the related work. Section 7 concludes the paper and outlines areas for future work.

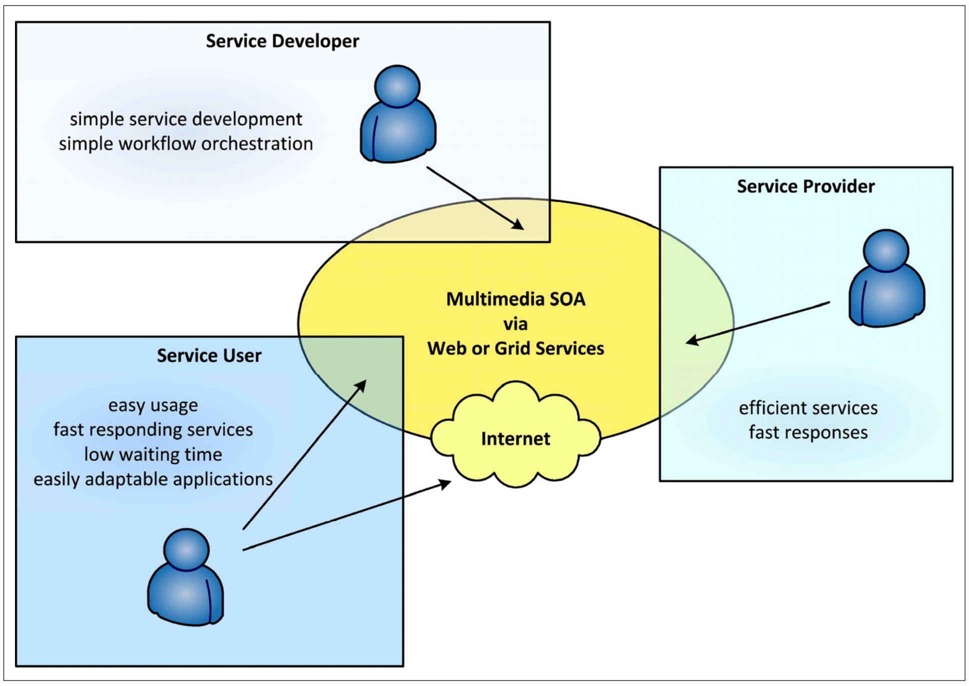
2 A service-oriented architecture for multimedia processing

At first glance, the SOA paradigm seems to be an appropriate solution for multimedia tasks. But to build such a *multimedia SOA* with a suitable set of tools and technologies, the general roles of an SOA, the structure of typical multimedia workflows and the implications of using BPEL, scalability issues and topics like ease of use should be taken into account.

2.1 General roles in a multimedia SOA

Many roles can be distinguished in a typical SOA (Kajko-Mattsson *et al.*, 2007). However, for an SOA for multimedia analysis, a coarse distinction between different roles is sufficient. Figure 1 shows a model of the general actors involved in a multimedia SOA, along with some of their requirements.

Figure 1 The general roles in a multimedia SOA (see online version for colours)



A *service developer* is interested in an easy way to develop new applications and services. He/She is assisted in this task by using additional middleware technologies for the transfer of large amounts of data that cannot be managed efficiently via SOAP. Furthermore, using a workflow editor, a service developer combines low-level services such as feature extraction and classification with new high-level services such as face detection or cut detection in videos, speaker recognition in audio streams and so on.

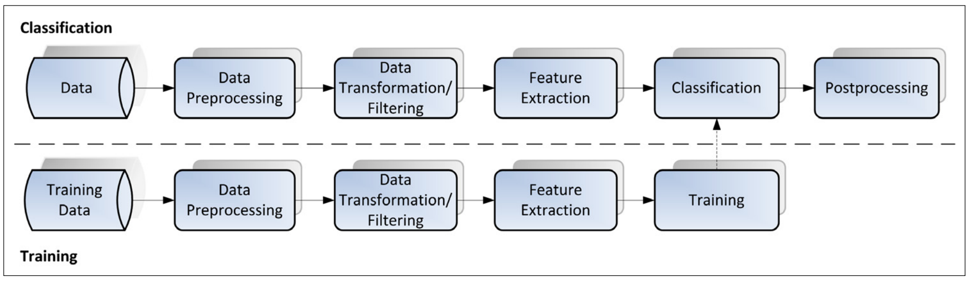
The *service provider* is interested in efficient services in that the response time is minimised for users. File transfers should be handled efficiently, thus reducing the response time of the service for the user.

The *service user* is interested in using services in an easy manner. For this reason, a browser able to handle the WSDL description of web services is useful. Some users might be interested in modifying their application or creating a new one. This can be achieved by a mash-up editor. A mash-up editor should be offered as an Rich Internet Application (RIA) based on widely spread technologies like Flash, JavaScript or other highly compatible technologies for users to avoid performing difficult installations.

2.2 Structure of a multimedia workflow

Most approaches for content analysis are built in a monolithic fashion following a sequential structure (see Figure 2 and Eide *et al.*, 2002) and are divided into two phases: training and classification. In general, preparatory tasks are executed to generate the smallest entities of interest needed to solve the problem in question, *e.g.*, retrieving images from a database, decoding video frames or decompressing audio signals. Next, some preprocessing is done, such as transforming or filtering. Then, the basic step of feature extraction will be performed, followed by the actual classification. Sometimes, a post-processing step is needed, *e.g.*, building a representation of the analysis results. The training phase is similar, but a training step instead of a classification step is performed.

Figure 2 The multimedia analysis procedure (see online version for colours)



Furthermore, the classification step can be seen as a feature extraction step offering the opportunity to combine it with other features for building more complex analysis approaches.

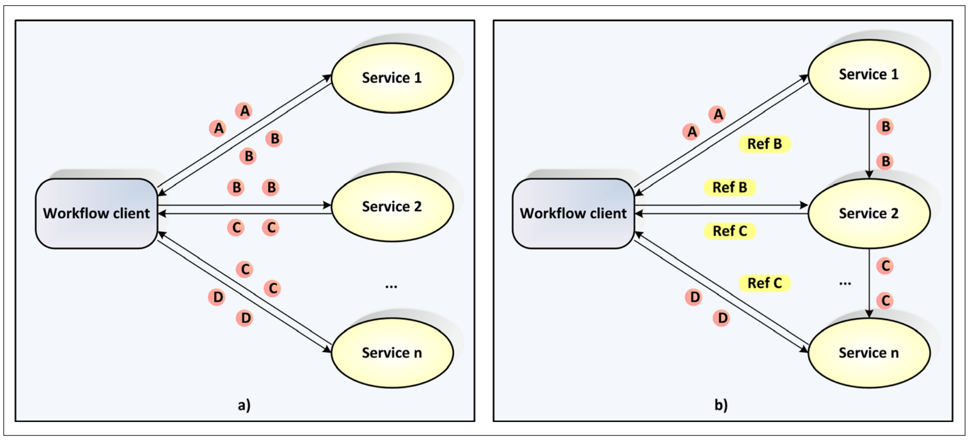
The different tasks in the sequential structure are candidates to be separated and wrapped as services. By allowing the composition or reconfiguration of the existing services, workflow languages speed up the development of distributed multimedia analysis applications.

2.3 Modelling the data flow in BPEL

BPEL as an orchestration language for business processes defines a workflow as a set of activities. These activities are divided into two groups: basic and structured. The basic activities describe single steps for interaction with a service and/or the manipulation of data passing the engine. With the help of structured activities, the order of activity execution is dictated. Thus, BPEL provides a rich set of elements to explicitly model the control flow (*i.e.*, sequence of invocations, fault handling, *etc.*), but the data flow is only modelled implicitly by the manipulation and assignment of data encapsulated in request/response messages. For example, the normal way of modelling the data exchange between two simple web services is as follows: invoke the first service with a request message, wait for the corresponding response from the service, assign all needed data to the input variable of the second service, invoke it with a request message encapsulating the data and wait for the response. In the context of highly frequent data exchange and transfer of large amounts of data, this way of modelling the data flow is not efficient.

Figure 3(a) shows the typical sequence of service invocations and the corresponding data flow. For example, the workflow engine invokes a preprocessing service and sends the multimedia data (A) to it. The service returns the results (B) to the workflow engine and these results are again transferred to one or more other services that perform the next step of multimedia analysis. Clearly, some of the data ((B) and (C)) are transferred at least twice. If, for example, Service 1 is a video splitter service, the video would be transferred to the video splitter and the video parts are moved back to the BPEL engine and then to different services. This induces a significant load on the BPEL engine.

Figure 3 The data flow via a BPEL engine (see online version for colours)



To reduce the load on the BPEL engine, references can be used to achieve efficient data transmissions in BPEL workflows. Figure 3(b) shows the data flow via *references* under the full control of the BPEL engine. A reference points to a resource location in memory or to a file. The service can then directly pull the data from the resource location. The data does not have to be sent to the BPEL engine; only the references have to be sent. Since the references are very small, the load on the BPEL engine is very small, too. Thus, the use of *references* circumvents to transfer data twice.

For the services to which data are sent repetitively (for example, video frames are sent repetitively), two ways to model the data flow are proposed in this paper:

1 For each resulting processing unit (video part, frame, audio sample, *etc*.), a reference passes the BPEL engine and is sent to the destination service. This way, the BPEL engine keeps full control over the data flow. Furthermore, the control flow resides at the engine. Thus, it is possible to model conditional elements in reaction to the responses sent by the services. Failure and compensation handling is also possible in the workflow.

2 For communication to each service, a reference to a memory location is generated and the data are 'streamed' from there. This way, it is possible to write to a memory location which the target service can read from. The BPEL engine is only able to select the services that should then communicate via the references, *i.e.*, BPEL builds the streaming chain and then eventually waits for a callback or control messages. Since even less data are transferred, this approach is expected to be very efficient.

2.4 Scalability

When using workflows, the end point of each service invocation is typically set statically (Andrews *et al.*, 2003). However, if the demand for an actual workflow increases (Elson and Howell, 2008), the service invocations on a particular machine increase as well. With this increased demand, it is possible that these systems become overloaded, resulting in the service not being available anymore. If no explicit fault handling is performed within the workflow, the latter one will abort and the computation time is wasted.

To keep services available, an easy way is to extend the (hardware) platform on which the services are executed. In this way, the variety of resources for a certain service is bigger and the load can be reduced to keep services available. This is done manually by starting machines, deploying the necessary software and, if necessary, adapting the workflow. But preferably, this should be done automatically.

A workflow engine should be used to track the load of the platform and, if the system gets overloaded, it should dynamically provide new machines and perform service invocations on these new machines. This can be done by utilising cluster or grid desktop resources (Tanaka *et al.*, 2009). Since it is unlikely that as many machines as needed are available in-house for this dynamic growth, the use of so-called cloud resources is possible. The EC2 is one candidate for such cloud resources and can dynamically provide new resources to be used for the platform to keep the system load at an acceptable quality level. It offers each user a set of virtual machines in which his or her applications are deployed.

Because the start-up (and shutdown, if the load decreases) of virtual machines are not performed by the EC2 itself (but by an external client), the workflow engine should handle these tasks.

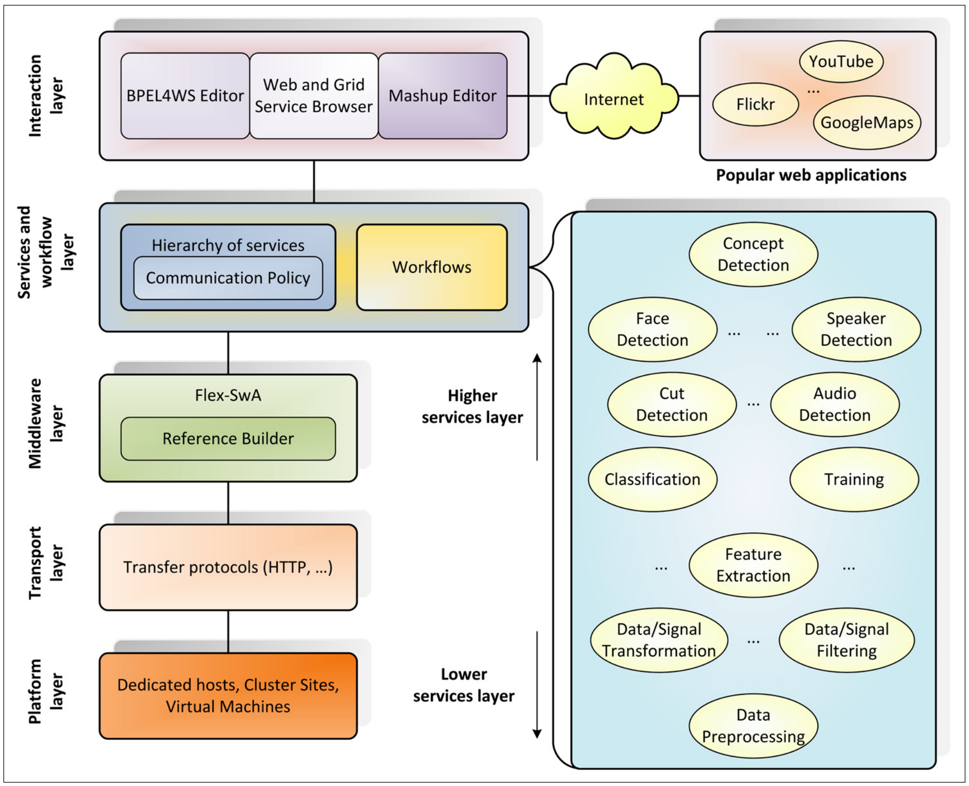
2.5 Ease of use

When offering web services in an SOA, there is often the problem that such services cannot be used easily. Client programs or portlets have to be written by developers and a portal has to be offered by the service provider. Most WSDL files miss a working and easy-to-use client to invoke web services. Potential web service users have to write their own client software to test and use a web service. Even if clients are available, they are often built for specific platforms, such as the Microsoft .NET framework. This limits their use to people using the Microsoft Windows operating system or computer experts who know how to compile or build their own client software. Therefore, familiar web-based
environments that can be installed easily, like a web browser or websites with server-side software based on widespread technologies like Hypertext Markup Language (HTML), JavaScript, Java and Flash, are desirable.

2.6 Multimedia SOA layer model

Taking into account the requirements of different actors, the restrictions imposed by modelling the data flow with BPEL, the structure of typical multimedia workflows and scalability and ease-of-use requirements, a layer model for a multimedia SOA (as shown in Figure 4) is proposed, consisting of five layers: interaction layer, services and workflow layer, middleware layer, transport layer and platform layer.

Figure 4 The layer model for a multimedia analysis framework (see online version for colours)



The *interaction layer* provides a front end to communicate with the multimedia SOA. A developer uses a BPEL workflow editor like the grid-enabled ViGO (see Dörnemann *et al.*, 2007) to define new workflows out of existing services and workflows from the services and workflow layer. Workflows and single services can be executed by the web and grid services browser (Heinzl *et al.*, 2008b) handling user interface generation and service execution. Service users may even create new applications by using a grid-enabled mash-up editor that allows the combination of a predefined set of services (*e.g.*, face detection) and web applications like Flickr and so on.

The *workflow layer* consists of services from the services layer and possibly, of other workflows. A workflow may already indicate a binding of services to actual compute nodes, *i.e.*, which compute nodes are used for which services.

The *services and workflow layer* consists of workflows and services arranged in an arbitrary number of sublayers. Workflows consist of services and possibly other workflows. The services sublayers range from low-level services such as feature extraction and feature classification over higher-level services such as face detection and cut detection to high-level services like actor detection (answering questions such as ‘How many minutes was Actor A on the screen?’). Services can be combined with new services across all sublayers. A communication policy (Heinzl *et al.*, 2008a) describes the protocols on which a service relies when data transfers or streaming capabilities are needed.

The *middleware layer* uses Flex-SwA (Heinzl *et al.*, 2006; Seiler *et al.*, 2008) to provide references to resources. These references are mainly used for streaming capabilities such as repetitive sending of frame data or when large files, *e.g.*, videos, have to be transferred, as SOAP is not well suited for both tasks. With the help of Flex-SwA, the proposed data flow modelling approaches are integrated into the multimedia SOA.

The *transport layer* realises the actual transmission of data from the interaction layer to services, between services on specific platforms and from services to the interaction layer.

The *platform layer* provides the resources needed to execute services. This includes dedicated and dynamic resources. The former ones are in-house desktop pools or cluster nodes. Dynamic resources can be virtual machines provided by Amazon EC2. A scalable infrastructure can be achieved using a dynamic provisioning component within the workflow engine, as proposed by Dörnemann *et al.* (2009a).

2.7 Flex-SwA

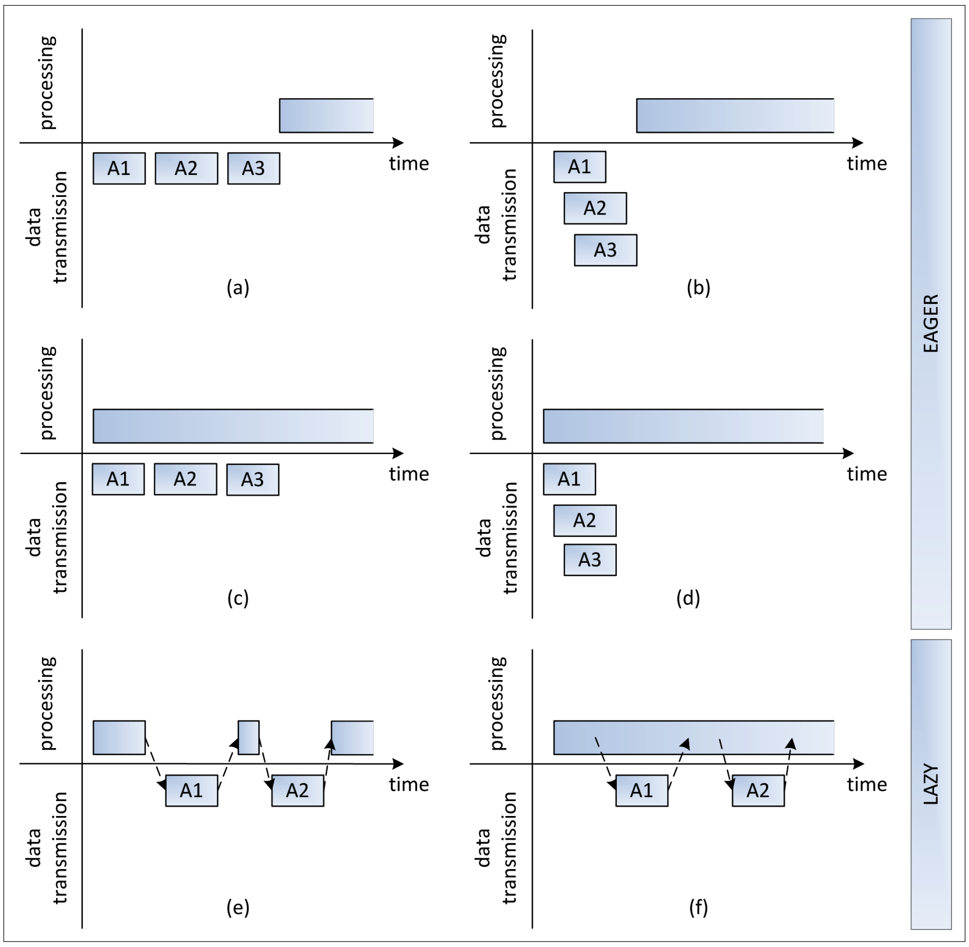
Flex-SwA (Heinzl *et al.*, 2006) uses references to flexibly and efficiently handle data transfers. The *references* in Flex-SwA point to data locations. For instance, these data locations can be files hosted by third-party servers or the client itself or buffers in the client's main memory or in the memory of third-party servers. The references to memory locations can, for example, be represented by a Uniform Resource Identifier (URI) containing a Universally Unique Identifier (UUID). Also, the repetitive sending of data is handled via references, such that a reference can be forwarded to the node(s) that should actually consume the data.

Flex-SwA also offers different behaviours related to the transmission of binary data and the execution of services. These are service execution patterns, data transmission patterns, concurrency patterns and blocking mode patterns. For each service, a combination of these behaviours can be chosen by the application developer.

- *Service execution patterns* – there are two possible behaviours regarding the handling of data transmission and the execution of a service. In the *non-overlapping* mode, the platform performs all data transfers prior to the invocation of the service; in the *overlapping* mode, data transmission and service execution are performed in parallel.
- *Data transmission patterns* – the platform can be instructed to perform an *eager* or *lazy* transmission of referenced data resources, meaning that references are resolved as soon as possible or only upon a real attempt to access their content. The latter is especially useful when a data resource can be omitted, *e.g.*, if an error has occurred.
- *Concurrency patterns* – data resources can be transmitted in an *iterative* or *concurrent* fashion. If data resources are transmitted iteratively, only one data resource at a time is retrieved. If the data resources are transmitted concurrently, all data resources are retrieved in parallel.
- *Blocking mode patterns* – two blocking mode patterns are offered: *blocking* and *nonblocking*. If a service is in blocking mode, it requests the retrieval of a data resource and waits until it is fully available. If a service is in nonblocking mode, service execution resumes directly after the retrieval request.

The reasonable combinations of data transmission, service execution, concurrency and blocking mode patterns are shown in Figure 5.

Figure 5 The communication patterns in Flex-SwA (see online version for colours)



A combination of non-overlapping transmission handling and the eager transmission mode (Figure 5a) results in the transfer of every data resource before the service starts. This scheme is similar to the transmission via SwA (see Barton *et al.*, 2000). The transmission of data resources can also be done concurrently (Figure 5b), for example, by using several threads, thus providing the possibility of improving the transfer rate. Combining overlapping and eager transmission handling (Figure 5c) results in the immediate start of data transmission and the service. This mode is useful if the service has a certain warm-up time or does not need any data resources at the service's start. Here again, a concurrent transmission of data resources (Figure 5d) possibly provides a better transfer rate than the iterative approach. Lazy data transmission in combination with overlapping transmission handling results in an on-demand transmission of data resources. If the service needs a data resource, transmission is triggered at that time. This can be done in a blocking manner (Figure 5e), *i.e.*, the service is blocked until data are retrieved from the remote source and stored locally by the infrastructure, or in a non-blocking manner (Figure 5f), *i.e.*, the service only triggers the transmission and continues directly; the transferred data may be accessed by the service upon reception. The blocking mode is used if the service needs the complete data resource before the service can resume execution. The non-blocking mode can be used if only a part of the data resource is needed by the service.

In addition to the communication patterns, a service can be configured to persist data as a file prior to accessing the data or to keep the data in memory. Direct memory access promises a faster processing of data, while persisting data has the advantage of huge amounts of data being handled better since disk space is available in larger amounts compared to the main memory.

2.8 ViGO

ViGO (see Dörnemann *et al.*, 2007) is a tool for graphical workflow creation. It has been especially designed (*i.e.*, web services according to the Web Service Resource Framework or WSRF). This tool complies with the BPEL 1.1 standard (Andrews *et al.*, 2003) and allows the use of ‘normal’ web services. When creating workflows, the user is assisted by several wizards, minimising the need to know the details of the BPEL syntax. Furthermore, all operations in the editor are designed such that the whole process remains consistent.

ViGO is based on the Domain-Adaptable Visual Orchestrator (DAVO) (Dörnemann *et al.*, 2009b). DAVO is a domain-adaptable, graphical BPEL workflow editor. The key benefits that distinguish DAVO from other graphical BPEL workflow editors are the adaptable data model and user interface that permit customisation to specific domain needs.

2.9 Web and grid service browser

The web and grid service browser (Heinzl *et al.*, 2008b) is a familiar entry point for end users to ‘enter’ and use a service-oriented grid and easily invoke web services. It extends the functionality of portals and renders WSDL files directly in addition to normal HTML files, such that a service developer does not need to develop a client anymore and a service provider does not need to host and maintain a portal. For the end user, a graphical front end is generated that eases the invocation of web and grid services or the submission of a job. A user does not need to create a client anymore. Data transfers are integrated into the service invocation using an implementation of the Flex-SwA architecture, such that a user does not need to know the location he/she has to transfer data to and does not need to learn how to use data transfer mechanisms like Secure Shell (SSH) or GridFTP. Furthermore, the web and grid service browser takes care of security and certificate handling.

2.10 Mash-up editor

The goal of the mash-up editor is to provide an easy-to-use front end for a predefined set of services of the multimedia SOA as well as for popular web applications such as YouTube, Flickr, GoogleMaps, *etc*. In contrast to other mash-up editors, this mash-up editor converges the use of web and grid services with popular web applications. The editor provides two views:

- an application developer view
- an application view.

In the `application developer view`, new applications can be defined and saved as new components. The components are saved in an Extensible Markup Language (XML) format from which Adobe Flex components and BPEL workflows can be generated. An application developer mostly handles specific parts of the output of one service or application and supplies other services or applications with it. He/She deals mainly with result sets, *i.e.*, with arrays or lists of one type of data like images, videos, *etc*. By connecting the inputs and outputs of existing components, new components can be generated.

Instead of working on a large scale (with arrays of images and videos), the user can simply use the mash-up editor in the `application view`, where he/she can work with concrete search results, single videos and images.

To invoke grid services, the capabilities of the web and grid service browser are used. More precisely, a remote execution engine that can be triggered via a Hypertext Transfer Protocol (HTTP) request handles the invocation of the grid service.

3 Services of the multimedia SOA

The goal of the multimedia SOA is to offer several analysis services and tools to easily use and recombine these services. Currently, the service offer comprises audio analysis services, video analysis services and consumer services.

1 *Audio analysis services*

- *Automatic speaker diarisation* – The service provides state-of-the-art techniques to perform speaker diarisation: for a given input audio or video file, an index is automatically created that tells ‘who spoke when’. This means finding all segments of speech and assigning those segments that have been spoken by the same speaker to the same cluster.
- *Feature extraction* – This service provides access to 21 different feature extraction methods, including well-known Mel-Frequency Cepstral Coefficients (MFCCs). Taking an audio or video file as input, the feature vectors according to the chosen method and parameter settings are returned.
- *Voice model building* – Acoustic models (dedicated, but not limited to speech) can be built using this service. The user can choose among all prevailing model types, including Gaussian mixture models, hidden Markov models or one-class support vector machines.

- *Audio resynthesis* – The single steps of an audio analysis process bear many sources of confusion for a potential user: the different methods for feature extraction or model building are complex in themselves and a great number of available parameters for each method makes it difficult – even for an expert – to tell what effect a certain choice may have on the final result. The service offers methods to resynthesise the results of a specific analysis step using a specific parameter setting. A user can then listen to the result and gain intuitive insight into the effects of his/her selections from the way it sounds. By resynthesis, we mean the synthesis of a sound from features or models (that have previously been built from an audio file, thus 're'-synthesis) in a way that strives not for a pleasing or natural sound, but for an audible representation that makes the settings of the feature/model perceivable.

2 *Video analysis services*

- *Decoder* – This service decompresses MPEG-1 and MPEG-2 video files and outputs single video frames.
- *Feature extractor* – This service computes the dissimilarity of consecutive video frames by means of colour histogram differences and motion-compensated pixel differences. These features are used by the shot boundary detector.
- *Shot boundary detector* – Temporal segmentation into meaningful units is an essential prerequisite for video indexing and retrieval purposes. Shots are separated in a video by abrupt shot changes (cuts) and gradual shot transitions such as fade and dissolve.
- *MP7 converter* – MPEG-7 (Salembier, 2002) is an ISO standard that formalises the description of metadata for multimedia documents. Basically, the MP7 service transforms analysis results (*e.g.*, the result of temporal video segmentation) into a description that complies with the MPEG-7 standard and the service outputs corresponding MP7 files. Furthermore, the service is capable of merging several MPEG-7 input files into a single MPEG-7 representation.
- *Video face detector* – This service detects frontal faces in a video frame. This service is based on Intel's Open Source Computer Vision Library (OpenCV) (Intel Corporation, 2008) implementation of the face detector that was originally proposed by Viola and Jones (2004).
- *Video splitter* – The video splitter splits MPEG-1 and MPEG-2 video files into MPEG-1 and MPEG-2 video parts.

3 *Consumer services*

- *Video on Demand (VOD)* – A user may invoke the VOD service to either subscribe to a VOD multicast based on Real-time Transport Protocol (RTP) (Schulzrinne *et al.*, 2003) or as a front end to YouTube to receive a streamed Flash video.

3.1 Use cases

For evaluation purposes, three use cases have been selected, one from each domain.

3.1.1 Audio resynthesis

For audio analysis, the audio resynthesis service called `WebVoice` is evaluated. It offers four different operations that make the intermediate result at specific points in the analysis process audible:

1 *wav2splice* – This operation produces a 'spliced' version of the input audio. The original signal is segmented into blocks of specifiable length and the time order of these blocks is then randomised. This helps in analysing the effect of time ordering on the perception of an audio signal.

2 *wav2features* – Here, MFCCs are extracted from the given audio signal based on the given parameter setting. The feature vectors are then resynthesised into a signal, helping the user understand the effects of different parameter settings like pre-emphasis, filter bank size or the number of coefficients.

3 *wav2gmm* – This is one of two operations to build a statistical model from audio features (that are automatically extracted from the given audio signal). It allows the configuration of the creation of a Gaussian mixture model with a full or diagonal covariance matrix. The result is a signal resynthesised from the feature vectors (MFCCs) drawn from the model according to the probability density function it represents. Listening to it evokes a sense of what the model represents in general.

4 *wav2hmm* – This operation uses a hidden Markov model for resynthesis, as stated above.

The evaluation of this use case will show that Flex-SwA is a more efficient and flexible technology for transferring bulk data (in this case, a wave file) than standard mechanisms like SwA and SOAP itself. The different communication patterns allow an investigation of the best configuration for data transfers that may also rely on the underlying hardware (*e.g.*, multicore or single-core processors). Furthermore, for interaction with the service, the functionality of the web and grid service browser as a tool dealing with web or grid services in general and multimedia services in particular is shown.

3.1.2 Video face detection

From the domain of video analysis, a simple workflow consisting of three services for face detection in videos is presented. Three variations of this workflow are discussed, each of them using a different style of data transmission.

A monolithic implementation for face detection typically consists of the following steps:

Step 1 An MPEG decoder sequentially produces a series of frames from a given input video.

Step 2 Each decoded frame is processed by a face detection algorithm mainly consisting of feature extraction and classification. The results of this algorithm are bounding rectangles stored in a simple list.

Step 3 After processing the whole input video in this manner, a final list is given to a component, which uses this information to build an MPEG-7 file containing meta-information for multimedia data.

This monolithic implementation can be decomposed by wrapping the functionality performed in the steps into services. A service-oriented realisation could, for example, cover three services: MPEG decoder, face detector and MPEG-7 converter. Three different workflows have been designed using these services, each with a different data transfer style:

1. The first workflow relies strictly on the exchange of SOAP messages. Decoded frames and detected faces are encapsulated in these messages and delegated by the orchestration engine to the face detection service and MPEG-7 converter service.
2. The second workflow uses Flex-SwA's reference concept; the data remain in the memory of the data producer (decoder or face detector) and only a small reference is transmitted in a SOAP message from the producer (via the orchestration engine) to the consumer (face detector or MPEG-7 converter), who can then pull the data directly.
3. The last workflow also uses Flex-SwA. Just one reference is delegated from the decoder over the engine to the face detector and another reference, from the face detector over the engine to the MPEG-7 converter. These two references are used to install a fixed connection between the services for frame and face data transmission, such that a service can repeatedly read the data directly from the producer.

The evaluation of this use case will show that the two proposed approaches to model the data flow explicitly by using Flex-SwA in BPEL workflows are significantly more efficient than modelling the data flow implicitly. One approach leaves full control of the data flow with the BPEL engine; the other approach shifts control of the data flow to Flex-SwA. The design of the three workflows has been alleviated by the ViGO workflow editor. The interaction between the user and the services takes place again by using the web and grid service browser.

3.1.3 Video on demand

An evaluation of a VOD service is presented from the area of consumer services. The integration of the VOD service will show the limits of web services with respect to streaming, soft real-time requirements and file transfers.

There are two main ways to organise the transmission of a video in a VOD application:

1. The video is completely transferred to the end user before playback is started. Thus, the user can skip parts of the video.
2. The video is streamed in 'real time' to the user or partially buffered. Thus, the user cannot jump forward to parts that have not yet received.

The first possibility is not very well suited if a user wants to watch a video *ad hoc*, since transferring the video may take a lot of time. Using SOAP even makes the situation worse. The second possibility is more suitable (and very popular, *e.g.*, YouTube) for watching videos in an *ad hoc* manner. If the video is sent to several customers, RTP will possibly be used for transmission. Another possibility is to use Transmission Control Protocol (TCP) for point-to-point communication and buffer parts of the video locally before watching the video. To integrate the VOD application into an SOA with standardised web service technologies, RTP packets have to be sent over SOAP.

The evaluation of this service shows that it is hard to achieve an SOAP-based VOD service with a sufficient performance. Three scenarios are presented, each showing a different degree of integration into the SOA. The first scenario presents a standard VOD application that is not integrated into the SOA. The second scenario presents a service that realises the RTP communication over SOAP. The third scenario shows how a communication policy (Heinzl *et al.*, 2008a) can be utilised to express the need for RTP as the communication protocol and integrate it into the SOA.

For interaction with the user, the Flash variant of the service is shown with the help of the mash-up editor.

3.2 Implementation issues

The implementation of the described services has been realised either as web or grid services. Web services are basically stateless, meaning that between two invocations, a web service does not remember information about its clients. Grid services, on the other hand, manage states according to the WSRF (see Graham *et al.*, 2006) and can provide information about their current progress, which is usable for long service runs. An advantage of web services over grid services is that clients can be built more easily for web services, since better tool support exists for this task.

The `WebVoice` service for audio resynthesis is based on the Speaker Classification Library (`SCLIB`). This C++ class library is currently under development at the University of Marburg, Germany, and provides state-of-the-art speaker recognition and related algorithms, methods and tools under a unified interface. The service has been implemented as a web service in Apache Axis 1.4 since it takes 'only' a plain 16 kHz, 16-bit mono-channel MS-RIFF WAV file as input and returns another wave file representing the selected features. Until now, the service works in a monolithic fashion, such that parallelisation is not yet possible. But since the implementation of the service is efficient, it might be sufficient to replicate the service.

The video face detection service has been realised by implementing several grid services in Java with the help of the 'Service Generator' of the Grid Development Tools (GDT) (see Friese *et al.*, 2006). As a grid middleware, the widespread Globus Toolkit (Foster, 2005) was used. Since the feature extraction task of the face detection algorithm is the most time-consuming task, it is desirable to parallelise that task. The use of grid services enables us to move these services to a virtualised cluster in the future.

The first grid service wraps an MPEG decoder written in C++; the second one is the video face detector that uses the algorithms of the OpenCV library (Intel Corporation, 2008) and the third one converts the analysis results of the preceding service into a standard MPEG-7 file. All services publish several methods to support different workflows.

The additional features that grid services offer also require a more complex implementation. Since the use of grid services is not directly supported by graphical workflow modelling tools in the BPEL environment, the BPEL-based workflows have been designed with ViGO, a BPEL editor able to design workflows consisting of grid services and a modified ActiveBPEL engine (Dörnemann *et al.*, 2007) able to handle grid service invocations and platform integration (*e.g.*, Amazon EC2).

The VOD service was implemented with help of the Java Media Framework, which provides an RTP implementation that can be used to stream videos over the network. The YouTube front end has been implemented as an Adobe Flex component.

4 Quantitative evaluation of the use cases

After describing the use cases and their implementation, a quantitative evaluation of the use cases is presented.

4.1 Audio resynthesis (WebVoice)

The `WebVoice` service was tested with different Flex-SwA communication and memory patterns to determine the most efficient pattern for the underlying hardware and operating system (dual-core Pentium D 3GHz processor with 3 GB of RAM and a Microsoft Windows XP Professional SP2 operating system). The client was executed on another computer with the same specifications on the same LAN. The implementation of the service may depend on the pattern used. If a lazy pattern is used, the implementation effort may increase. Listing 1 shows the implementation of the `wav2splice` operation of the `WebVoice` service used for all communication and memory patterns except for lazy nonblocking.

Listing 1 Implementation of the `wav2splice` operation (exception handling omitted)

```
public Reference [] wav2splice (Reference [] ref,
  boolean usePhaseSynthesis, int preservedBlockSize,
  int intermediateFrameCount, double steepness,
  double olaErrorTarget, int olaMaxIterationCount)
{
  Reference [] res = new Reference [ref. length];
  for (int i = 0; i < ref. length; i++)
  {
    InputStream is = ref[i].acquire ();
    SC_Wrapper wrapper = new SC_Wrapper();
    String fName = wrapper.wav2splice (is,
      "fb12_webvoice/webvoice.ini",
      FlexSwAConstants.getFileServerDir(), usePhaseSynthesis,
      preservedBlockSize, intermediateFrameCount, steepness,
      olaErrorTarget, olaMaxIterationCount);
    res[i] = returnReferenceFromString(fName);
    is.close();
    return res;
  }
}
```

The `wav2splice` operation of the `WebVoice` service takes an array of references and the parameters for the actual C++ `wav2splice` function. The command `ref[i].acquire()` loads the referenced data either from the local disk or memory or from the remote machine to the local disk or memory, depending on the communication or memory pattern specified. For the lazy nonblocking variant of the service, the implementation is shown in Listing 2.

The loading of the referenced data is initiated for all files before the analysis of the first wave file starts. Since the nonblocking pattern is used for the service, the `ref[i].acquire()` command does not block. Before a wave file is given to the `SCLIB`, asking the `SemaphoreRegistry` ensures that the data referenced by the `reference` object are completely transferred, which is a requirement for the `SCLIB`.

Listing 2 Implementation of the lazy nonblocking variant of the `wav2splice` operation (exception handling omitted)

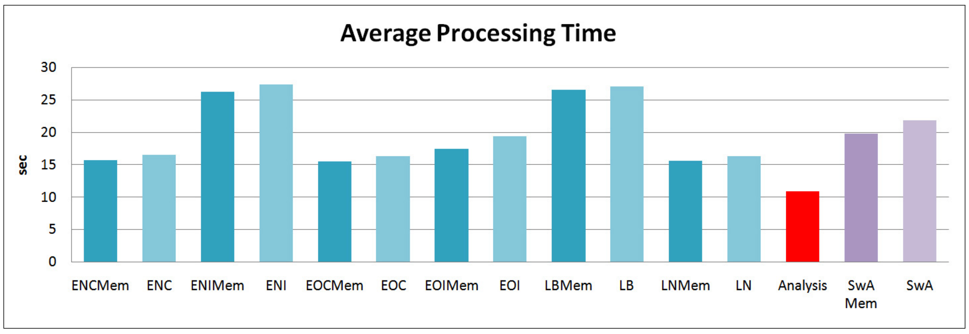
```
public Reference[] wav2splice(Reference[] ref,
  boolean usePhaseSynthesis, int preservedBlockSize,
  int intermediateFrameCount, double steepness,
  double olaErrorTarget, int olaMaxIterationCount)
{
  Reference[] res = newReference[ref.length];
  InputStream[] is = new InputStream[ref.length];
  for (int i = 0; i < ref.length; i++)
  {
    is[i] = ref[i].acquire();
  }
  for (int i = 0; i < ref.length; i++)
  {
    SemaphoreRegistry.getInstance().acquire(ref[i].getUrid());
    SC_Wrapper wrapper = new SC_Wrapper();
    String fName = wrapper.wav2splice(is[i],
      "fb12_webvoice/webvoice.ini",
      FlexSwAConstants.getFileServerDir(), usePhaseSynthesis,
      preservedBlockSize, intermediateFrameCount, steepness,
      olaErrorTarget, olaMaxIterationCount);
    res[i] = returnReferenceFromString(fName);
    is[i].close();
    return res;
  }
}
```

As a reference implementation, an SwA implementation of the `wav2splice` operation was used, as shown in Listing 3.

In the SwA implementation, the attachments have to be detached from the SOAP message (if needed) by iterating over the attachment parts of the SOAP message. From each attachment part, the input stream is given to the `wav2splice` C++ function.

The average processing time of the service was measured for 50 runs from the client's machine. The measurements consist of transferring five wave files, each with a size of 9.5 MB, to the service, analysing the files and returning a reference to the spliced files. The red bar (Analysis) in Figure 6 indicates the time needed to analyse these five audio files. The other bars additionally show the time of the data transfers (*i.e.*, analysis *and* data transfers) for different communication patterns and SwA.

Figure 6 The average processing time of the `wav2splice` operation (see online version for colours)



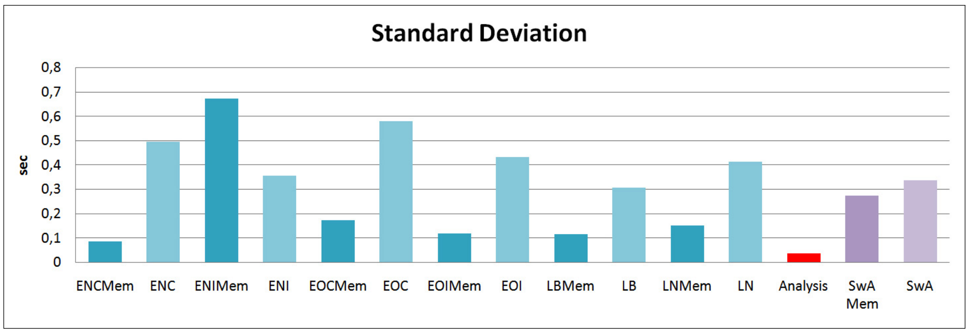
The analysis of the five audio files took 10.87 *sec* on average. When using SwA for data transfer, the time for analysis and data transfers was 19.82 *sec* when directly handling the content of the SOAP message (SwA Mem) or 21.88 *sec* when the received attachments are first been written to the local disk and then processed by the `SCLIB` (SwA). When using one of the concurrent memory modes, the processing time speeds up to 15.62 *sec* (ENC mem), 15.51 *sec* (EOC mem) and 15.60 *sec* (LN mem). The concurrent persistent modes are a little slower: 16.51 *sec* (ENC), 16.33 *sec* (EOC) and 16.31 *sec* (LN). The modes where the data transfer is done iteratively before the analysis are slower compared to SwA because five different connections are used to transfer the data: 26.25 *sec* (ENI mem), 26.55 *sec* (LB mem), 27.35 *sec* (ENI) and 27.06 *sec* (LB). Although the EOI mem and EOI modes are iterative modes, they perform a little faster than SwA and SwA mem due to the overlapping of service execution and data transmission. After the first file arrives, analysis and the data transfers are executed concurrently. Generally, when handling the data in memory, the processing is a little faster.

Listing 3 Implementation of the `wav2splice` operation with SwA (exception handling omitted)

```
public Reference [] wav2splice (boolean usePhaseSynthesis,
  int preservedBlockSize, int intermediateFrameCount,
  double steepness, double olaErrorTarget, int olaMaxIterationCount)
{
  int i = 0;
  Message m = MessageContext.getCurrentContext().getCurrentMessage();
  Reference[] res = new Reference[m.countAttachments()];
  Iterator<AttachmentPart> it = m.getAttachments();
  while (it.hasNext())
  {
    AttachmentPart a = it.next();
    InputStream in = (InputStream) a.getContent();
    SC_Wrapper wrapper = new SC_Wrapper();
    String fName = wrapper.wav2splice(in,
    "fb12_webvoice/webvoice.ini",
    FlexSwAConstants.getFileServerDir(),
    usePhaseSynthesis, preservedBlockSize, intermediateFrameCount,
    steepness, olaErrorTarget, olaMaxIterationCount);
    res[i] = returnReferenceFromString(fName);
    i++;
  }
  in.close();
  return res;
}
```

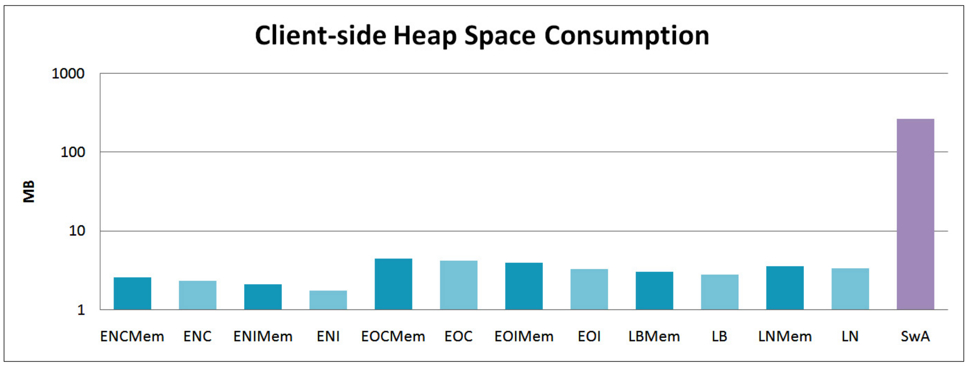
Figure 7 shows the standard deviation; it is below 0.7 *sec* for all experiments.

Figure 7 The standard deviation to invoke the `wav2splice` operation (see online version for colours)



The memory footprint at the client's side is lower when using Flex-SwA, which is an additional advantage. The client-side memory usage for the different patterns and SwA is shown in Figure 8. While the `WebVoice` Flex-SwA client does not need more than 4.41 *MB* of heap size for the different patterns, the `WebVoice` SwA client already needs 268.67 *MB*. The problem is even more severe for larger files; this is due to the implementation of SwA in Apache Axis. But the problem of an attachment being loaded to memory as a whole on the client or server's side persists.

Figure 8 Client-side heap space consumption (see online version for colours)



To summarise, the evaluation of Flex-SwA communication and memory patterns exemplified by an audio resynthesis service has shown that selecting the correct communication patterns can speed up the processing time of the service by either overlapping data transfers and service execution or handling data transfers concurrently. In addition, the use of Flex-SwA reduces the development effort compared to SwA.

4.2 Face detection in videos

For the face detection service, four different scenarios are investigated and compared with the performance of the monolithic implementation to determine the overhead of using ActiveBPEL and Globus Toolkit and of distributing this scenario within a LAN.

In the first scenario, the *original* implementation of the video analysis running on a dual-core Pentium D 3GHz processor with 3 GB of RAM and a Microsoft Windows XP Professional SP2 operating system is tested. In the second scenario, the different video analysis services are deployed to the Globus service container and executed by the ActiveBPEL engine. The ActiveBPEL engine and Globus service container run on the same computer. For the third scenario, four computers are used. The ActiveBPEL engine runs on a Core 2 Duo processor with 3GHz and 2 GB RAM and a Fedora 9 operating system. The Globus middleware is distributed to three computers, each with a dual-core Pentium D 3GHz processor with 3 GB of RAM and a Microsoft Windows XP Professional SP2 operating system. The computers are connected by a 100 Mbit/s Fast Ethernet network.

For the second and third scenario, the total time of the analysis is measured:

- when decoded frames and detected faces are sent in SOAP messages
- when a Flex-SwA reference is sent for *each* decoded frame and detected face
- when data are streamed directly without passing the ActiveBPEL engine at all.

Normally, SwA would be used for binary data instead of normal SOAP, but SwA is not supported by Globus Toolkit 4.0.x. Because the sending of uncompressed frame information as an array in SOAP messages leads to large messages of approximately 5 MB, all array structures are transformed to simple strings, leading to a size of approximately 1 MB.

A fourth scenario tests how much time the face detection service needs on the Amazon EC2. Normal SOAP messages are sent in this scenario.

In the first three scenarios, two videos from the TRECVID 2006 test set were analysed. In the last scenario, only the second TRECVID video was tested. Each analysis was run 20 times. The results of the first three analyses (consisting of the average of the measured total times) are shown in Figures 9 and 10. In all scenarios, the video is transferred to the first service before the service is invoked and the measurement begins.

The total time of the original implementation is used as a reference for the other scenarios. Figure 9 shows the results for the first video and Figure 10 shows the results for the second video. In the diagrams, the time measurements for four different local implementations are presented. The first local implementation is the original monolithic implementation; analysing the first video took approximately 180 sec and analysing the second video took 244 sec (188/244). Analysing the same videos using SOAP took approximately 490/631 sec. When using references, the analysis speeded up (219/286 sec) compared to the SOAP version. Using streaming capabilities is even slightly faster than the original implementation: the analysis took approximately 170/225 sec. While the original monolithic implementation is not multithreaded, the services are capable of using several threads leveraging the parallel processing capability provided by the dual-core processors, thus resulting in a faster execution of the analysis.

Figure 9 The analysis runtimes for video 20051227_125800_CNN_LIVEFROM_ENG.mpg (see online version for colours)

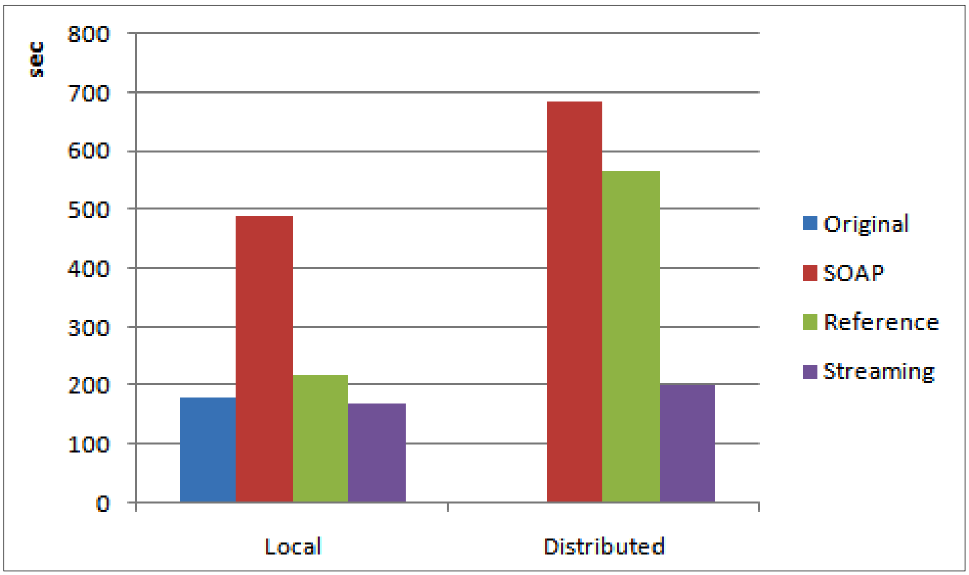
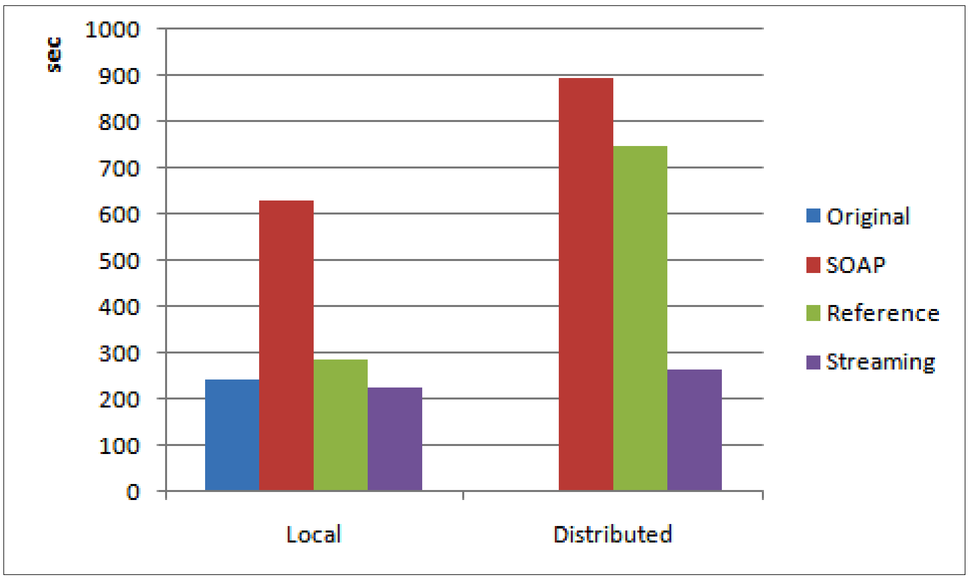


Figure 10 The analysis runtimes for video 20051209_125800_CNN_LIVEFROM_ENG.mpg (see online version for colours)



In tests with a distributed environment, each service is placed on a different computer. Data transmission via SOAP messages took approximately 683/896 sec. The references approach speeded up the total time of the distributed analysis to 570/749 sec. The distributed streaming approach, however, is the fastest approach; it is nearly as fast as the local streaming approach and needs only 200/264 sec.

In the distributed environment, the reference approach is slower than the streaming approach since a SOAP message has to be sent and processed for each frame, whereas in the streaming approach, only one SOAP message has to be sent and processed.

The two proposed approaches based on the extended Flex-SwA framework outperform normal SOAP messages. The streaming approach is much faster than the other approaches at the price of breaking the normal orchestration approach. The communication is shifted to the services or Flex-SwA framework, respectively, bypassing the BPEL engine. However, the workflow engine can – with additional implementation effort – control the data flow via callbacks.

For applications that do not necessarily need the best possible runtime behaviour, the proposed reference approach is a suitable compromise between good runtime behaviour and leaving full control at the workflow engine. Furthermore, it is possible to send several frames via one reference, resulting in a tradeoff between data flow control and efficiency. This way, a developer may gradually shift from full data flow control to more efficiency.

To get a comparison between the execution on physical hosts and virtual machines, analysis was also performed in a virtualised environment, namely Amazon EC2. The start of a virtual machine within the EC2 needs a special image, a so-called Amazon Machine Image (AMI), that has to be prepared beforehand. Such an image contains the operating system and all the necessary user libraries and codes. In the presented case, this means that an Ubuntu-based Linux operating system was used and the necessary native code and middleware had to be deployed.

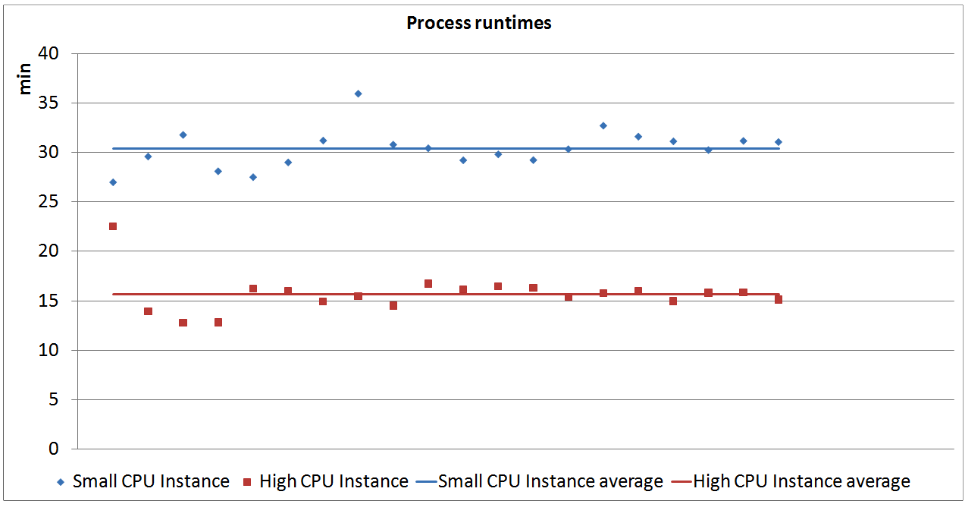
The ActiveBPEL workflow engine allows the use of a special `InvokeHandler` to process service invocations. The solution proposed by Dörnemann *et al.* (2009a) utilises this mechanism to obtain a scalable infrastructure. For this aim, a service is assumed to be reachable via a prepared virtual machine. Listing 4 shows the corresponding PartnerLink of the decoder when the dynamic scheduler is used. It can be seen that all the necessary information for the EC2 (like *AccessID* and *SecretKey*) is passed to the `InvokeHandler`. When an invocation is performed on this particular PartnerLink, the load balancer checks all the hosts offering that special portType for their load and performs the actual invocation on the host with the least load. If no such host is available, a new virtual machine within the EC2 is started and invocation is carried out there.

Listing 4 The dynamic InvokeHandler within the BPEL process

```
<partnerLink name = "decoderPL">
 <partnerRole endpointReference = "dynamic"
  invokeHandler = "java:LoadBalancer
  ?threshold = 1.0;accessID=***;secretKey=***;
  imageID = ami-95cc28fc; availZone = us-east-1c"/>
```

Amazon EC2 uses a ‘pay-as-you-go’ model to price its resources. The storage, *e.g.*, for the AMI, is priced by the Amazon Simple Storage Service (Amazon Web Services LLC, 2009b) so that each GB costs $0.15. Computing power is priced per hour; the actual amount depends on the so-called instance type. Different instance types describe different machine set-ups. In the presented measurements, only 'small instances'[1] and 'high-CPU medium instances'[2] are used. The former costs $0.10 and the latter, $0.20 per hour.

Figure 11 The process runtimes in a virtual environment (see online version for colours)



As before, a video from the TRECVID 2006 test (20051209_125800_CNN_LIVEFROM _ENG.mpg) was used, and the messages were exchanged using pure SOAP. It can be seen in Figure 11 that when the small instance type was used, analysing the video took around 1823.82 sec, whereas when the high-CPU instance type was used, the analysis only took 941.29 sec. In comparison to Figure 10, the slowdown is around a factor of 2.04 for the small instance type and around a factor of 1.05 for the high-CPU instance type. Since the algorithms are single-threaded and the code runs in a virtual machine, this slowdown is acceptable. However, the advantage of a dynamically scalable infrastructure redeems this small performance decrease.

4.3 Video on demand

The VOD service was evaluated in three different scenarios, as described in Section 3.1. In the first scenario, a video and audio stream is transmitted from and to the local host via RTP. The time to pack and unpack the data was measured on a computer with an Intel Centrino processor with a 1.5GHz clock speed and 2GB RAM. Windows XP SP2 was the operating system. The RTP packets were sent via User Datagram Protocol (UDP). At the sender, the time for packaging an RTP packet to a UDP datagram and initiate the transmission was measured. At the receiver, the multimedia stream was read and shown to the user. Here, the time to receive the datagram and extract the RTP packet from the datagram was measured. The transmission of the video lasted 30 sec and was repeated five times.

In the second scenario, the sender transmits each RTP packet over SOAP, *i.e.*, every RTP packet is embedded into a SOAP message and then transmitted. Again, the time for packaging and unpackaging measured, including SOAP processing. The SOAP messages were sent via UDP. The Apache Axis serialiser and deserialiser were extracted from the Axis package so that their capabilities could be used without using the SOAP engine as a whole. The sender serialises the RTP data with the Axis serialiser as Base64Binary and then sends the SOAP message to the receiving end point as a UDP datagram. The receiver reads the datagram and then deserialises the RTP packet from the SOAP message. The time measured is expected to be higher since serialisation and deserialisation take place before transmission and after reception. Again, the transmission of the video lasted 30 sec and was repeated five times.

Table 1 summarises the results of these measurements. When the RTP communication is encapsulated over SOAP, the time for serialisation induces an overhead of a factor of approximately 10 compared to the preparation time. The decapsulation process even introduces an overhead of a factor of 1000 and more. The standard deviation for one of the runs was especially high at approximately 113 ms. Since a VOD application is a real-time application; outliers like this may not be neglected.

Table 1 The average time and standard deviation to prepare the transmission and reception of a multimedia stream

Transfer method	*Time for 'packaging' (in ms)*	*Time for 'unpackaging' (in ms)*
RTP	0.1331–0.136 avg	0.0098–0.0137 avg
	0.0436–0.077 stdev	0.0072–0.0497 stdev
RTP over SOAP	1.571–1.664 avg	16.3163–18.9898 avg
	11.1056–11.947 stdev	27.096–113.1239 stdev

When comparing the quality of the video-watching experience, it was noticeable that the video – when transmitted via SOAP messages – got stuck sometimes, even though it was not even transmitted over the network, so that network latencies can be excluded as a cause. Furthermore, the standard deviation of the receiving process shows that the time for deserialisation varies more strongly than in the normal RTP case. Hence, when realising RTP communication over SOAP, the quality of a VOD service is not adequate for end users. One conclusion is that this type of application is simply not suitable for web service technologies. Nevertheless, it makes sense to leverage the advantages of web services like selfdescribing interfaces, loose coupling, *etc.*, for this application.

This problem can be solved by relying on the protocols especially designed for this type of application, like RTP. When using a communication policy, as done in the third scenario, it is possible to declare which protocols an application depends on.

In the third scenario, it is shown how the VOD service can be integrated into an SOA using a communication policy. Instead of handling RTP communication over SOAP, the communication policy describes the protocols suitable for such an application.

A simple VOD web service that starts a video transmission when invoked has been developed. It returns the end point of the multimedia stream to the client. The client uses the end point to receive the multimedia stream. An excerpt of the service's WSDL with an embedded communication policy is shown in Listing 5.

Listing 5 The policy embedded in WSDL for a simple VOD service

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
 xmlns:cp="http://fb12.de/2007/05/communicationpolicy"...>
 <wsp:Policy>
  <wsp:All>
   <wsp:ExactlyOne>
    <cp:protocol name="rtp" protocolID="RTPv2"
     operationref="streamVideo"/>
   </wsp:ExactlyOne>
  </wsp:All>
 </wsp:Policy>
 <wsdl:message name="streamVideoResponse">
  <wsdl:part name="streamVideoReturn" type="xsd:string"/>
 </wsdl:message>
 <wsdl:message name="streamVideoRequest">
  <wsdl:part name="videoname" type="xsd:string"/>
 </wsdl:message>
 <wsdl:portType name="VideoOnDemand">
  <wsdl:operation name="streamVideo">
   parameterOrder ="videoname"
   <wsdl:input message="impl:streamVideoRequest"
    name="streamVideoRequest"/>
   <wsdl:output message="impl:streamVideoResponse"
    name=streamVideoResponse"/>
  </wsdl:operation>
 </wsdl:portType>
 ...
</wsdl:definitions>
```

The policy shows that the service will use RTP and that the client has to support this protocol to invoke the service.

The time needed in the third scenario for RTP communication is in the same dimension as the times of the first scenario, since only the pure packaging and unpackaging times have been measured. The service invocation itself is the only overhead compared to using the application without services. However, this could be anticipated since communication is still realised via RTP.

Thus, using this communication policy allows the integration of services with soft real-time requirements into the multimedia SOA, which could otherwise hardly be integrated. The communication policy provides a standard way for these services to describe protocol needs for file transfers, streaming and real-time.

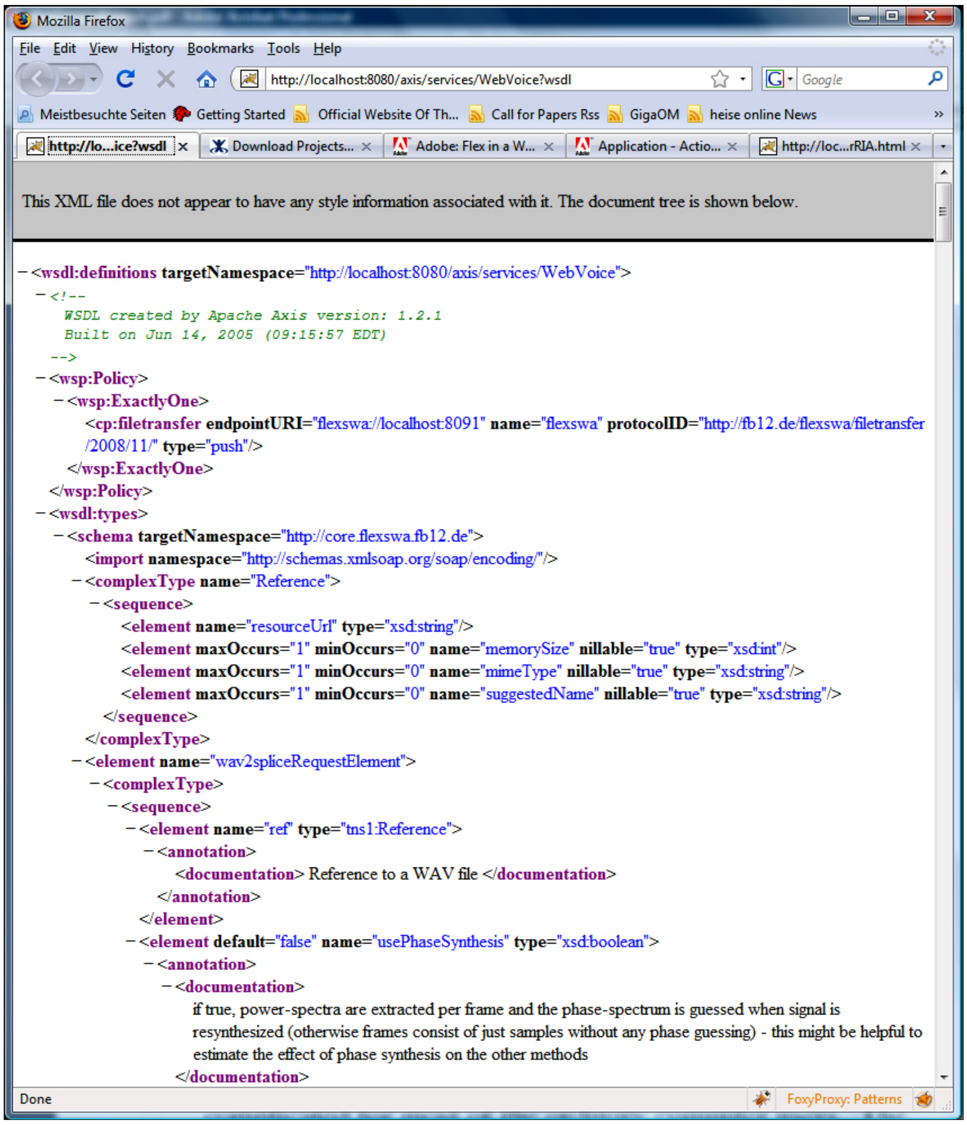
5 Qualitative evaluation of the use cases

5.1 Audio resynthesis (WebVoice)

To achieve a user-friendly but also generic interface for services, the web and grid service browser (realised as a Firefox plugin) is used. The web and grid service browser automatically generates a user interface when browsing to the WSDL of a service. It handles security configuration and proxy certificate generation when invoking grid services and helps users provide relevant information to invoke services; it also takes care of service invocation itself according to style/use combinations. For multimedia data, the web and grid service browser provides special result presentations to make audio files audible and visualise images and videos.

When normally browsing to a WSDL of a service, an XML tree appears, as shown in Figure 12.

Figure 12 A screenshot of Mozilla Firefox showing a WSDL file (see online version for colours)

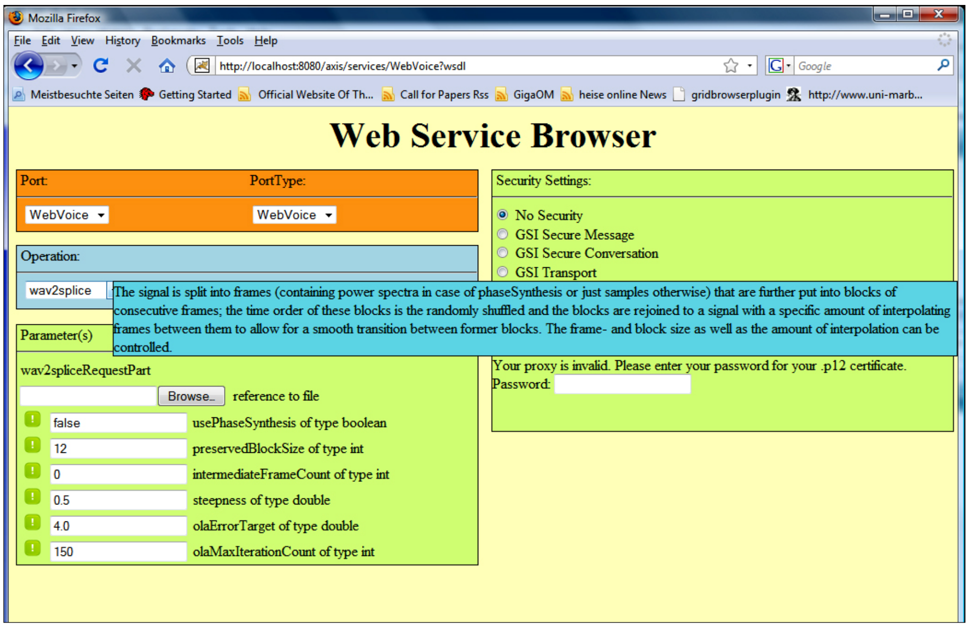


This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <wsdl:definitions targetNamespace="http://localhost:8080/axis/services/WebVoice">
  - <!--
      WSDL created by Apache Axis version: 1.2.1
      Built on Jun 14, 2005 (09:15:57 EDT)
    -->
  - <wsp:Policy>
    - <wsp:ExactlyOne>
        <cp:filetransfer endpointURI="flexswa://localhost:8091" name="flexswa" protocolID="http://fb12.de/flexswa/filetransfer
        /2008/11/" type="push"/>
      </wsp:ExactlyOne>
    </wsp:Policy>
  - <wsdl:types>
    - <schema targetNamespace="http://core.flexswa.fb12.de">
        <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      - <complexType name="Reference">
        - <sequence>
            <element name="resourceUrl" type="xsd:string"/>
            <element maxOccurs="1" minOccurs="0" name="memorySize" nillable="true" type="xsd:int"/>
            <element maxOccurs="1" minOccurs="0" name="mimeType" nillable="true" type="xsd:string"/>
            <element maxOccurs="1" minOccurs="0" name="suggestedName" nillable="true" type="xsd:string"/>
          </sequence>
        </complexType>
      - <element name="wav2spliceRequestElement">
        - <complexType>
          - <sequence>
            - <element name="ref" type="tns1:Reference">
              - <annotation>
                  <documentation> Reference to a WAV file </documentation>
                </annotation>
              </element>
            - <element default="false" name="usePhaseSynthesis" type="xsd:boolean">
              - <annotation>
                - <documentation>
                    if true, power-spectra are extracted per frame and the phase-spectrum is guessed when signal is
                    resynthesized (otherwise frames consist of just samples without any phase guessing) - this might be helpful to
                    estimate the effect of phase synthesis on the other methods
                  </documentation>
```

With the WSDL alone, a user without a computer science background will have severe problems invoking a service. The web and grid service browser, however, automatically generates a user interface when the user browses to the WSDL description of the service. The user interface generated when browsing to the `WebVoice` service's WSDL description is shown in Figure 13.

Figure 13 A screenshot of the `WebVoice` service user interface (see online version for colours)



Web Service Browser

Port: WebVoice
PortType: WebVoice

Security Settings:
- [✓ No Security]
- GSI Secure Message
- GSI Secure Conversation
- GSI Transport

Operation: wav2splice

The signal is split into frames (containing power spectra in case of phaseSynthesis or just samples otherwise) that are further put into blocks of consecutive frames; the time order of these blocks is the randomly shuffled and the blocks are rejoined to a signal with a specific amount of interpolating frames between them to allow for a smooth transition between former blocks. The frame- and block size as well as the amount of interpolation can be controlled.

Parameter(s)

wav2spliceRequestPart

	reference to file
false	usePhaseSynthesis of type boolean
12	preservedBlockSize of type int
0	intermediateFrameCount of type int
0.5	steepness of type double
4.0	olaErrorTarget of type double
150	olaMaxIterationCount of type int

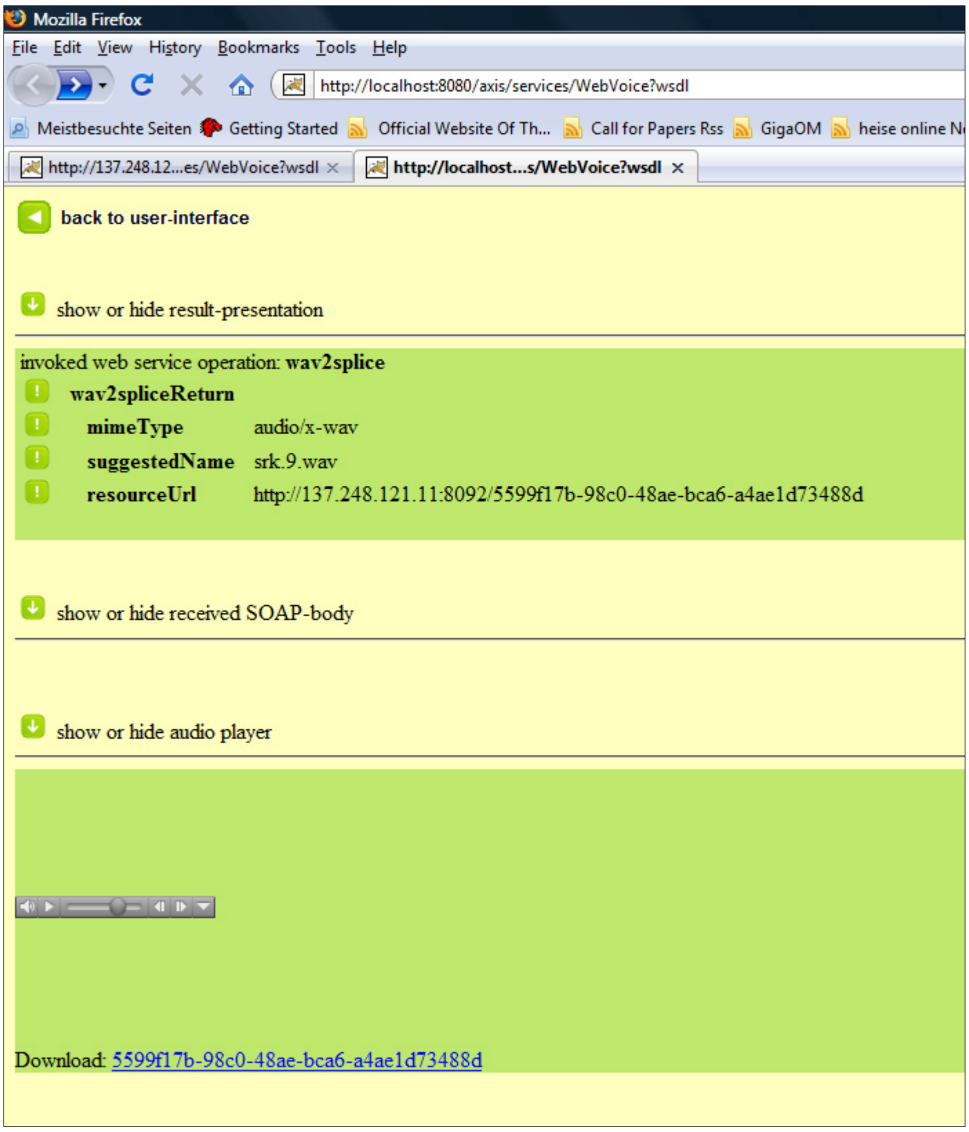
Your proxy is invalid. Please enter your password for your .p12 certificate.
Password:

When the user slides over the drop-down operations menu, a tooltip helps in choosing the right operation. The form fields are already filled with the default values set in the XML schema part of the WSDL file. To transfer an audio file, the user pushes the browse button and selects a file from the hard disk. The audio file is then transferred to the web service via Flex-SwA. The data are pushed to the service when the user is in a private network. If the user has a public Internet Protocol (IP) address, the local file is streamed to the service, such that the processing of the data can overlap with transmission. After the analysis, the result presentation engine generates an HTML page giving the user the choice of downloading or listening to the resynthesised audio (see Figure 14).

In the first section of the results page, the data of the resulting SOAP message are shown. When the mouse slides over the white exclamation mark, the data type is shown as a tooltip. The second section is reserved for the original SOAP message (hidden in the screenshot). By clicking on the green 'down' arrow, sections can be hidden and shown again. In the third section, a media player is embedded into the results page that plays the resynthesised audio file.

Therefore, normal computer users can simply invoke web services found on the web, especially web services processing and offering multimedia content.

Figure 14 The result of the `wav2splice` operation (see online version for colours)


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/axis/services/WebVoice?wsdl
Meistbesuchte Seiten
Getting Started
Official Website Of Th...
Call for Papers Rss
GigaOM
heise online N
http://137.248.12...es/WebVoice?wsdl
http://localhost...s/WebVoice?wsdl

back to user-interface

show or hide result-presentation

invoked web service operation: **wav2splice**

wav2spliceReturn

mimeType	audio/x-wav
suggestedName	srk.9.wav
resourceUrl	http://137.248.121.11:8092/5599f17b-98c0-48ae-bca6-a4ae1d73488d

show or hide received SOAP-body

show or hide audio player

Download: 5599f17b-98c0-48ae-bca6-a4ae1d73488d

5.2 *Face detection in videos*

The workflow built for face detection was already exposed as a web service for quantitative evaluation. To enable a user to invoke the service easily, it is again reasonable to use the web and grid service browser. The user interface to invoke the service is similar to the user interface of the `WebVoice` service. The results presentation, however, looks different; an array of images is returned with the detected faces highlighted according to the MP7 file. The images are automatically visualised by the web and grid services browser, as shown in Figure 15.

Figure 15 A screenshot of the results of the face detection workflow service for a Meat Loaf music video (see online version for colours)

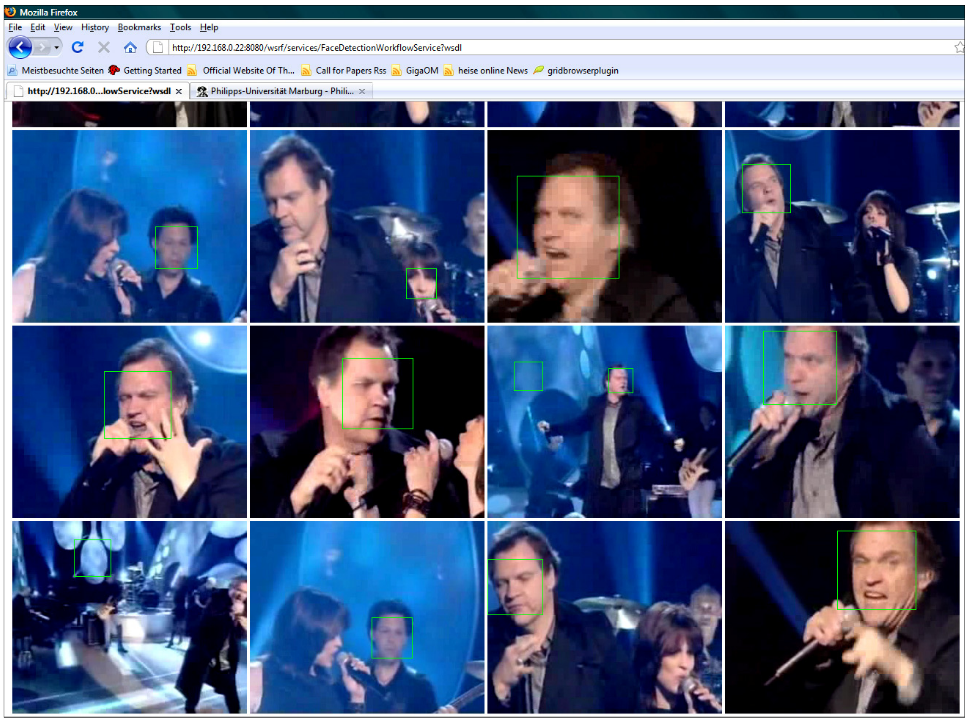
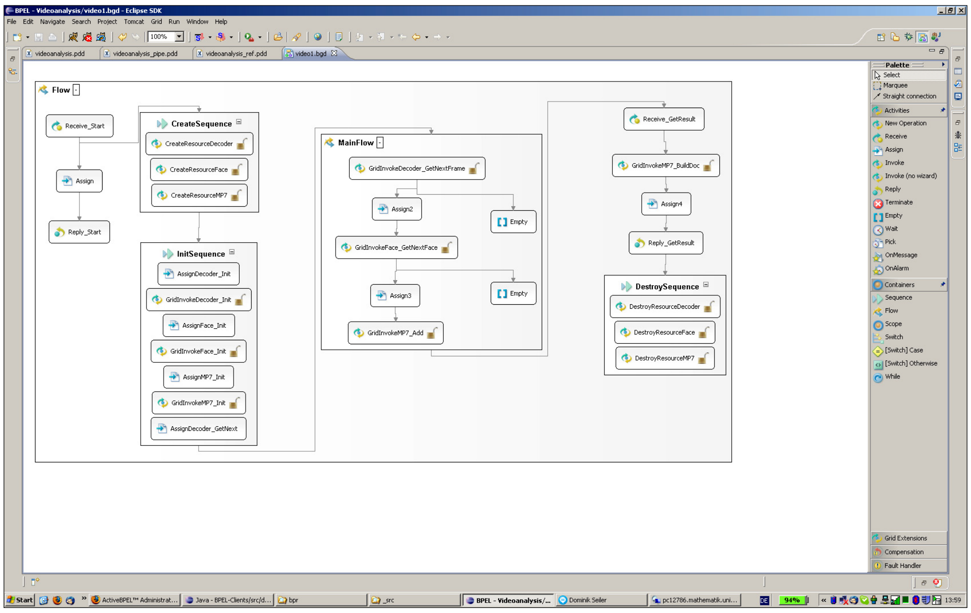

Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.0.22:8080/wsrf/services/FaceDetectionWorkflowService?wsdl
Meistbesuchte Seiten Getting Started Official Website Of Th... Call for Papers Rss GigaOM heise online News gridbrowserplugin
http://192.168.0....lowService?wsdl
Philipps-Universität Marburg - Phili...

Figure 16 A screenshot of the face detection workflow in ViGO (see online version for colours)



For the developer, the work is eased by using ViGO to design the workflow, as shown in Figure 16. When creating the workflow, the user is guided by wizards to add all the necessary invocations. Afterwards, the user has to add all assign operations, model the data transfer and create the links between activities to express the control flow.

The use of ViGO in tandem with the Grid-enabled workflow engine leads to considerable simplifications in the development and the treatment of Grid workflows. Grid services can be handled like classic web services, making no difference neither in the development nor in the execution.

Because of the inherent startup of virtual machines, the use of the load balancer even allows to achieve a scalable infrastructure.

5.3 Video on demand

For interaction with the Flash-based VOD streaming, our first prototype of the mash-up editor was used. The mash-up editor was created with Adobe Flex. A Flex component (shown in Figure 17) was built as an interface for YouTube, which allows clicking on an image preview of the search results to play this video in a player component, as shown in Figure 18.

A user can easily use the YouTube component to watch videos inside the mash-up editor. The next step is to provide components to convert the FLV video format to MPEG video and then use the video face detector service.

Figure 17 A screenshot of the Flex YouTube component (see online version for colours)

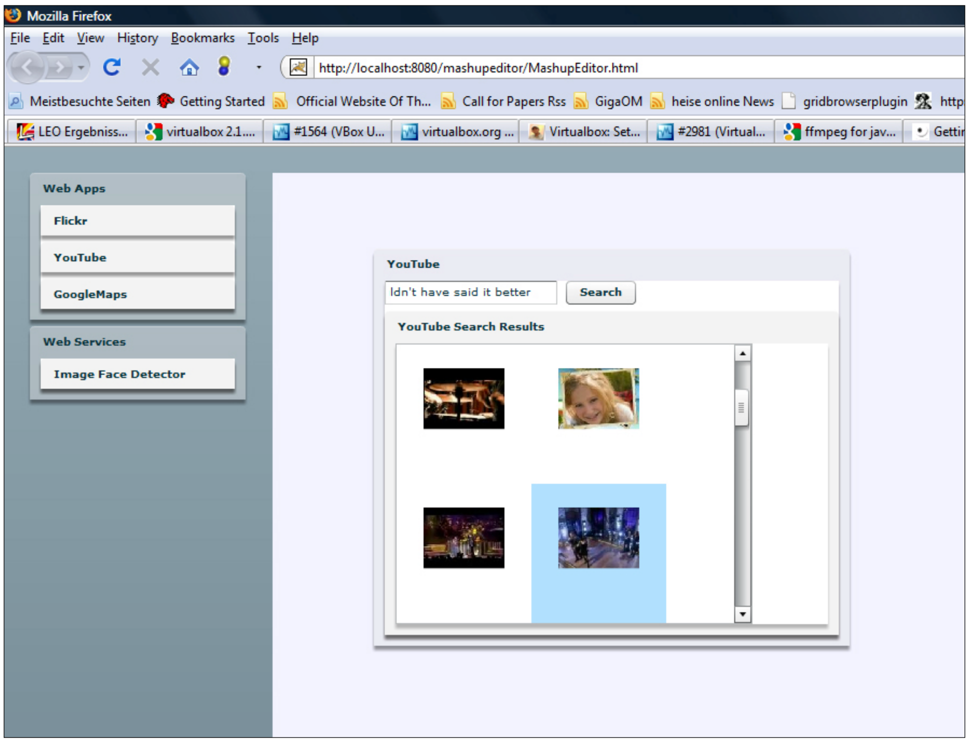
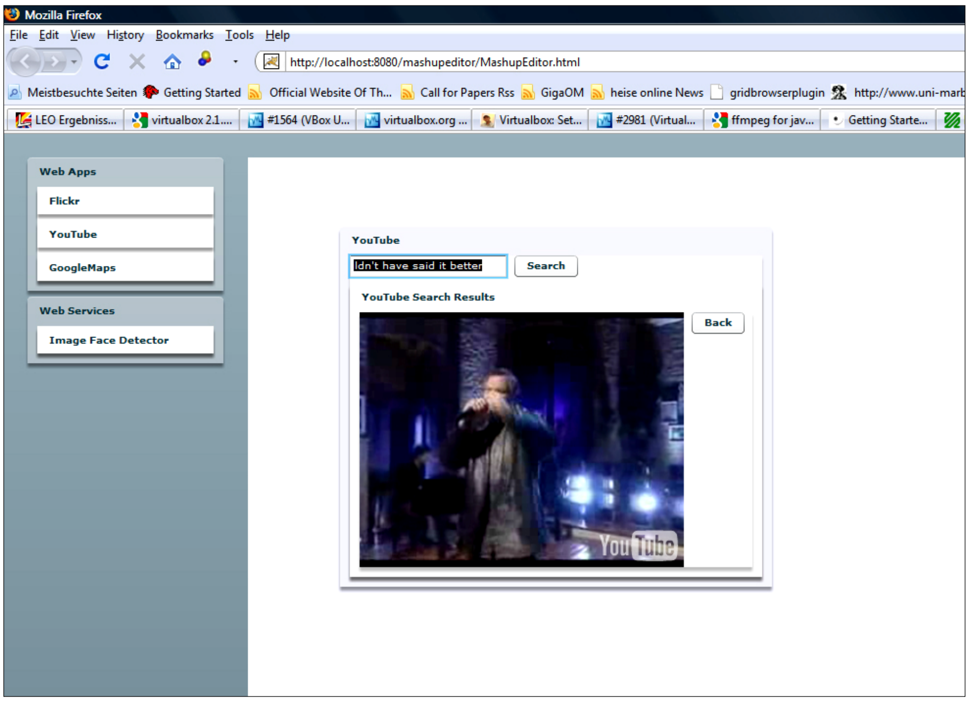

Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mashupeditor/MashupEditor.html
Meistbesuchte Seiten Getting Started Official Website Of Th... Call for Papers Rss GigaOM heise online News gridbrowserplugin http
LEO Ergebniss... virtualbox 2.1.... #1564 (VBox U... virtualbox.org ... Virtualbox: Set... #2981 (Virtual... ffmpeg for jav... Gettir
Web Apps
Flickr
YouTube
GoogleMaps
Web Services
Image Face Detector
YouTube
ldn't have said it better
Search
YouTube Search Results

Figure 18 A screenshot of the Flex YouTube player component (see online version for colours)


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mashupeditor/MashupEditor.html
Web Apps
Flickr
YouTube
GoogleMaps
Web Services
Image Face Detector
YouTube
ldn't have said it better
Search
YouTube Search Results
Back

6 Related work

In this section, an overview of the related work is given. First, other work on service-oriented video analysis is reviewed. Then, the existing approaches considering the issue of decentralised data transmission in centralised workflows are presented. Furthermore, other browser-based approaches to invoke web and grid services are presented and the Web 2.0 applications and mash-up editors that act as a front end for SOAs are summarised. Finally, the related work dealing with dynamic resource provisioning using BPEL is presented.

6.1 Service-oriented video analysis

The CASSANDRA framework (Nesvadba *et al.*, 2005) is a distributed multimedia content analysis system based on an SOA and aims to facilitate the composition of applications from the distributed components on a network of cooperating devices (like personal computers or consumer electronics equipment). The individual analysis components are encapsulated into functional units called 'service units'. All units on one particular device are managed by a local component repository synchronised by a master repository. Service composition is initiated by a special coordination component. Each service unit has a control and data-streaming interface. The control interface is based on the Universal Plug and Play (UPnP) framework, while the streaming interface is based on TCP/IP. This framework does not use web services to build an SOA neither for the service definition nor for workflow composition; it uses UPnP. Since service composition is performed by a special component, the abovementioned coordination component is just used to set up the workflow. In general, this framework is similar to our streaming approach, but does not leverage the expressiveness of BPEL or the flexibility of Flex-SwA. Data transport is fixed to TCP/IP, whereas the use of Flex-SwA allows the use of custom protocols, *e.g.*, Reliable File Transfer (RFT), GridFTP, HTTP or UDP. The use of UPnP may be useful for controlling consumer electronics, but is critical in terms of security and will probably not find a broad adoption by network administrators.

Barker *et al.* (2008) presented an architecture and an Application Programming Interface (API) to support a distributed data flow model in conjunction with a centralised control flow. To provide a noncentralised data flow from one service to another, the authors made use of proxies that are responsible for data management and service invocation. The proxies are placed near the service that they should administrate. A service invocation has to be passed over a proxy, the produced data are saved through the proxy and a reference to this data is delegated back to the calling centralised workflow engine. This solution is just focused on WSDL-based web services and does not directly support WSRF-based services, which are often more suited for multimedia content analysis tasks. Although this approach leads to a distributed data flow with reduced data movement, it also introduces an additional indirection stage for service calls. Since the proxies should be placed on the same web server or domain and the reference system needs to store the service results on a local disk, a resource competition between service and proxies may occur.

Zeng and Miao (2006) presented an approach to avoid a centralised data transmission in BPEL-based workflows in terms of grid computing. They also distinguished between intraworkflow and inte-workflow data communication. While the first model focuses on direct data transmission between two services in one workflow, the second model also covers data transmission between several grid workflows. Both models make use of two special grid services. In the case of intraworkflow communication, a so-called Data Transmission Factory Service (DTFS) is called from the workflow engine to set up a Data Transmission Service (DFS) that handles direct data transfer from one grid service to another and notifies the engine regarding its status. The workflow engine just handles control messages and small data packages. To support this kind of data transmission through BPEL, a new language element called `<dataTransmission>` has been defined. It specifies the source and target service and transport type, which can be GridFTP or notifications. In the interworkflow case, a so-called Data Proxy Service (DPS) is used to manage data transport between different grid workflows and/or services. The DPS is part of the workflow engine and has the same lifetime as the corresponding workflow. In contrast to the previous approach, this proposal is specialised for grid services and does not support ‘normal’ web services. Since BPEL is extended with an additional element, it does not conform to the standard; workflows cannot be executed on standard BPEL engines. Furthermore, in the case of interworkflow data transfers, a specially adopted workflow engine must be used for DPS instances.

Blower *et al.* (2006) developed a system for creating a new service type called Styx Grid Service (SGS). The system wraps command-line programs and allows them to be run over the internet. It is based on a Java implementation of the Styx file-sharing protocol and allows data streaming from service to service over transport protocols like TCP/IP or UDP. An SGS can be used in a web service or WSRF environment. An orchestration into workflows is also possible through simple shell scripts or a graphical workflow system like the Taverna workbench.[3] In contrast to SGS, our architecture is based on the *de facto* standard for business workflows (BPEL) and can be integrated into business processes. The granularity of an SGS is very coarse, since it can only wrap whole executables. Our approach works at the web or grid service level, that is, on methods or functions.

6.2 Familiar environment for simple service use

To simply use services in a familiar environment, a browser-based service invocation is reasonable.

Gemstone (Bhatia *et al.*, 2006) is an application based on the Mozilla Application Framework (Mozilla, 2009a) and XULRunner (Mozilla, 2009b) that allows users to browse a set of web services and enables the dynamic integration of user interface elements. The service providers have to specify the service and user interface. Gemstone is used to select services from a proprietary repository; hence, Gemstone lacks a real browser integration. It only provides the integration of web services for which visualisation code is written in XML User Interface Language (XUL) and JavaScript. This limits the use of Gemstone to service providers offering visualisation code to their services and using the proprietary repository format supported by Gemstone. Since Gemstone is built on top of the Mozilla Application Framework, it only supports Transport Layer Security (TLS) and PKCS12 certificates. The lack of proxy certificate support considerably limits its use in grid environments. Gemstone strictly supports client-side processing. For slow clients, it would be better if an option for server-side processing was available.

Web and grid portals like The GridSphere Project (2009) provide access to a collection of services and a set of tools such as single sign-on, data management and certificate management or collaboration capabilities (sharing, interlinking and integrating multidisciplinary data sets) like the GEON portal (Nambia *et al.*, 2006) directly through the browser. However, to operate portals, maintenance and administration efforts are needed. Furthermore, for each new service to add to the portal, a portlet has to be written.

The Web Services Remote Portlet (WSRP) specification (Kropp *et al.*, 2003) addresses a part of the problem. A user interface defined at a remote site can be included in the portal. Still, each provider has to define the user interface for each service description on the web or in a repository.

The Virtual Resource Browser (VBrowser) (Olabarriaga *et al.*, 2006) tries to make the grid more accessible to end users. It allows easy access to a number of high-level services like RFT, a job submission service, image analysis, *etc*. The VBrowser is a stand-alone application and is not integrated into a browser. But more importantly, it cannot be used to invoke services that have not been integrated into the application. Whereas the web and grid service browser generates the user interface on the fly, the VBrowser can only access the services that programmers have integrated programmatically into the browser. Therefore, the web and grid service browser is much more adaptable.

Table 2 shows a detailed classification of the capabilities of the proposed web and grid service browser, portals and Gemstone for several criteria.

Table 2 An overview of the capabilities of the browser extension, portals and Gemstone

Criterion	*Grid browser Extension*	*Portals (like GridSphere)*	*Gemstone*
Integration	Direct browser integration	Direct browser integration	New application based on Mozilla XULRunner
Installation	Installation is as simple as installing a Firefox extension	Complete Globus Toolkit installation, including MyProxy server and GridFTP portlet	Installation is as simple as installing a web browser
Ease of use	Arbitrary WSDL file is used to generate GUI	Each service will be integrated into the portal	WSDL is usable only if in a proprietary repository format *and* XUL and JavaScript have been written for it
Functionality	Grid and web services	Grid and web services	Web services
Processing	Local and server-side processing are possible	Server-side processing	Local processing
Security level	GSI support	GSI support	Only HTTPS
GridFTP support	Can be added by the Firefox extension TOPAZ	Yes	The Firefox extension TOPAZ can be used separately from Firefox
Certificates	Proxy certificates	Proxy certificates	'Ordinary' certificates

The capabilities of the web and grid service browser, portals and Gemstone can be classified by several criteria:

- *Integration* – The web and grid service browser as well as portals like GridSphere provide direct integration into the web browser, whereas Gemstone is a new application based on Mozilla XULRunner.
- *Installation* – The installation of the web and grid service browser is as easy as installing Firefox extensions, namely via drag-and-drop. A portal needs a completely installed Globus Toolkit including a *MyProxy* server and the GridFTP portlet. The complete Globus Toolkit requires a Unix operating system. The installation of Gemstone is comparable to the installation of a web browser.
- *Ease of use* – The web and grid service browser browses the location of an arbitrary WSDL file to generate the Graphical User Interface (GUI) via which the service described by the WSDL document can be invoked, whereas for each service to be integrated into the portal, a portlet and client have to be written. In Gemstone, only WSDL documents can be used if they have been stored in a proprietary repository format *and* the application developer has deposited XUL and JavaScript code, which is then executed by the Mozilla application.
- *Functionality* – The web and grid service browser and portals like GridSphere support grid and web services, whereas Gemstone only supports web services.

- *Processing* – This can take place either locally on the client or on the service side. The web and grid service browser supports both modes depending on whether local or remote user interface generators, execution engines and results presentation engines are used. Portals only support service-side processing; Gemstone supports local processing.
- *Security level* – The web and grid service browser and portals like GridSphere support the Grid Security Infrastructure (GSI). Gemstone only supports TLS via Hypertext Transfer Protocol Secure (HTTPS).
- *GridFTP support* – Portals like GridSphere support GridFTP. Gemstone and the web and grid service browser need the TOPAZ Firefox extension to add GridFTP functionality.
- *Certificates* – Portals like GridSphere and the web and grid service browser are able to use proxy certificates that are usually utilised when invoking a secured grid service. Gemstone only makes use of ordinary certificates.

Although the web and grid service browser has many advantages, it also has a few disadvantages. The web and grid service browser depends on the Firefox releases of the Mozilla Foundation. Furthermore, complex user interfaces might be better when they are tailored to the application than automatically generated.

6.3 Mash-up editors in the SOA context

Until now, only few research efforts have been undertaken in the context of mash-ups and web service-based SOAs. Approaches based on EzWeb/Fast by Lizcano *et al.* (2008) deal with, among others, the adaptability of business processes to changing environmental situations in the context of mash-ups. But their solution can only deal with web services based on the Representational State Transfer (REST) architectural style. This is not sufficient for integration into existing business processes. Our approach can be seamlessly integrated into existing business processing thanks to the use of BPEL.

Nestler (2008) proposed the use of mash-ups by means of communication between services and users. Furthermore, simple mash-ups that can be defined by end users are clearly separated from the complex development of workflows and business process integration. Nevertheless, the existing services of an SOA should be usable as part of mash-ups. Besides web services, our mash-up editor also supports grid services for computationally demanding tasks.

6.4 Dynamic resource provisioning using BPEL

Di Penta *et al.* (2006) presented a dynamic binding framework called WS-Binder. It allows the (re)binding of *partnerLinks* to services during the runtime of a process. For this aim, the authors used a proxy architecture. Based on a given policy, the framework is able to schedule an invocation to a specific service. This binding can either be done before or during the execution of the workflow. In addition, runtime recovery can be supported. However, the framework does not provide support to extend the pool of usable services, for example, by booting a virtual machine.

TRAP/BPEL, presented by Ezenwoye and Sadjadi (2007), is another framework for the dynamic adaptation of service compositions. It makes use of a generic proxy pattern. The proxy is able to query a Universal Description Discovery and Integration (UDDI) registry to find a suitable service. The binding is performed in an autonomic fashion; for example, if a service call fails, a substitute is determined and the call is retried. Furthermore, the selection of a suitable service can depend on a policy. To make a BPEL process interact with the TRAP/BPEL framework, it has to be adapted. Although policies can be used for service selection, there is no possibility to specify post-invocation policies, *e.g.*, for fault handling. Furthermore, the dynamic scaling of the infrastructure is not supported.

By introducing a new element (*find_bind*) into the BPEL language, Karastoyanova *et al.* (2005) presented their approach for runtime adaptability. The mechanism is able to find services, *e.g.*, by querying a UDDI registry. Based on policies, it selects suitable services and binds them to process instances. In case a service call fails, a process instance repair is guaranteed by rebinding to another port. Selection criteria can be modified at runtime. There is no support for dynamically providing additional target hosts.

Ma *et al.* (2008) presented a grid-enabled workflow management system based on BPEL. The system allows interaction with stateful resources, dynamic service binding and the scalability of workflow execution. In the scenario presented in the paper, the BPEL engine is a bottleneck. Scalability is achieved by placing a load balancer in front of a cluster of BPEL engines. Calls to the workflow engine are then scheduled with respect to the engines' workload. Dynamic service binding at runtime is achieved similar to our approach. At runtime, a provisioning service is contacted that looks up a host where the requested service is installed. However, the approach does not take the workload of target hosts into account. Furthermore, it does not provide any feature to dynamically provide additional target hosts.

7 Conclusions

In this paper, we presented a service-oriented infrastructure for multimedia processing that uses BPEL. Hence, the developed multimedia services can be easily integrated into existing business processes. To efficiently model the data flow in BPEL, Flex-SwA's reference concept is used. With the presented web and grid service browser and mash-up editor, users can easily use the services of the SOA and partially combine existing services and popular web applications to new services. To provide a scalable infrastructure, services can dynamically be added by leveraging Amazon's EC2. By using ViGO, service developers can easily create new services by combining existing ones. A communication policy allows services to express protocol requirements for file transfers, streaming or soft real-time. The evaluation of three use cases demonstrated the feasibility of our proposal.

Flex-SwA provides more efficiency for data transfers than SOAP and SwA when configured with the correct communication patterns. It was shown that the overlapping of service execution and data transmission as well as the concurrent handling of data transfers lead to better performance. Flex-SwA is well suited for the transfer of bulk data in multimedia environments, as demonstrated for the audio resynthesis service.

The streaming capabilities of Flex-SwA speeded up the analysis. Two approaches to model the data transmission of services were described. In the reference approach, a service returns a SOAP message with a reference to the calling BPEL workflow instance that then delegates this message to the destined consuming services. The consumer can then pull the data directly from the producer. This modelling style reduces the amount of data delegated via the workflow engine from one service to another. The streaming approach uses the Flex-SwA reference system to build up communication channels directly between a set of services. In contrast to the first modelling style, the task of data exchange is totally decoupled from the BPEL workflow. The BPEL engine just chooses the partners who will build a communication channel between them. This leads to a tradeoff between fast execution and control of the flow. Some control might be regained via callbacks.

For scalability reasons, services can be 'outsourced' to the Amazon EC2. Our evaluation has shown that the high-CPU instances of EC2 provide a similar performance to the regular computers we used for the analysis. Hence, it is not too hard to calculate how many additional hosts are needed when the load increases.

The web and grid service browser provides a familiar and simple user interface for service users. The user interface is automatically generated when the user browses to a WSDL file. Furthermore, the web and grid service browser helps fill out the generated forms, handles proxy certificate generation, handles GSI security and takes care of service invocation and result presentation. For multimedia data, special result presentations have been integrated into the browser to offer human users an easy way to see or hear results.

The evaluation of the VOD service has shown that it is hard to integrate certain types of applications into an SOA because of real-time constraints, streaming requirements or large file transfers. In the case of the VOD application, sending RTP packages over SOAP is just too inefficient to allow a user to still watch the video. The delay between the packages becomes too big. With the help of a communication policy, the requirements of the service can be described and the services can be integrated into an SOA. For interaction with the user, the mash-up editor provides a YouTube component as a front end to the Youtube website. With this component, videos can be searched, selected and watched.

There are several areas for future work. The next components to be added to the mash-up editor are a Flash-to-MPEG converter and the face detector service to allow an analysis of YouTube videos. For the face detection service, the suitable number of frames per reference to attain a good balance between data flow control and efficiency will be investigated. Furthermore, more tests of the WebVoice service on different hardware will be performed. These tests will help evaluate which communication patterns are especially useful for which hardware (dual core, single core, *etc*.). Finally, more tests with Amazon EC2 will be performed to make the transition from dedicated resources to cloud resources easier.

Acknowledgements

This work is partially supported by the German Ministry of Education and Research (BMBF) (D-Grid Initiative) and the Deutsche Forschungsgemeinschaft (DFG, SFB/FK 615).

References

Amazon Web Services LLC (2009a) 'Amazon Elastic Compute Cloud (EC2)', http://aws.amazon.com/ec2/.

Amazon Web Services LLC (2009b) 'Amazon Simple Storage Service (S3)', http://aws.amazon.com/s3/.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., *et al.* (2003) 'Business Process Execution Language for Web Services Version 1.1', http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

Barker, A., Weissman, J.B. and van Hemert, J. (2008) 'Orchestrating data-centric workflows', *CCGRID*, IEEE Computer Society, pp.210–217.

Barton, J.J., Thatte, S. and Nielsen, H.F. (2000) 'SOAP messages with attachments', *W3C Note*, http://www.w3.org/TR/SOAP-attachments.

Bhatia, K., Stearn, B., Taufer, M., Zamudio, R. and Catarino, D. (2006) 'Extending grid protocols onto the desktop using the Mozilla framework', *2nd International Workshop on Grid Computing Environments (GCE 2006)*, pp.1–8.

Blower, J.D., Harrison, A.B. and Haines, K. (2006) 'Styx grid services: lightweight, easy-to-use middleware for scientific workflows', *International Conference on Computational Science*, Vol. 3, pp.996–1003.

Di Penta, M., Esposito, R., Villani, M.L., Codato, R., Colombo, M. and Nitto, E.D. (2006) 'WS binder: a framework to enable dynamic binding of composite web services', *Proceedings of the 2006 International Workshop on Service-oriented Software Engineering*, ACM, pp.74–80.

Dörnemann, T., Friese, T., Herdt, S., Juhnke, E. and Freisleben, B. (2007) 'Grid workflow modelling using grid-specific BPEL extensions', *Proceedings of German e-Science Conference (GES)*, pp.1–8.

Dörnemann, T., Juhnke, E. and Freisleben, B. (2009a) 'On-demand resource provisioning for workflows using amazon's elastic compute cloud', *Proceedings of the 9th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '09)*, IEEE Press, to appear.

Dörnemann, T., Mathes, M., Schwarzkopf, R., Juhnke, E. and Freisleben, B. (2009b) 'DAVO: a domain-adaptable, visual BPEL4WS orchestrator', *Proceedings of the IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA '09)*, IEEE Press, to appear.

Eide, V.S., Eliassen, F., Granmo, O.C. and Lysne, O. (2002) 'Scalable independent multi-level distribution in multimedia content analysis', *Proceedings of the Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems (IDMS/PROMS)*, London, UK: Springer-Verlag, pp.37–48.

Elson, J. and Howell, J. (2008) 'Handling flash crowds from your garage', *ATC'08: USENIX 2008 Annual Technical Conference*, Berkeley, CA: USENIX Association, pp.171–184.

Ezenwoye, O. and Sadjadi, S.M. (2007) 'TRAP/BPEL: a framework for dynamic adaptation of composite services', *Proceedings of the International Conference on Web Information Systems and Technologies (WEBIST 2007).*

Foster, I. (2005) 'Globus Toolkit Version 4: software for service-oriented systems', in H. Jin, D.A. Reed and W. Jiang (Eds.) *International Conference on Network and Parallel Computing (IFIP)*, Lecture Notes in Computer Science (Springer-Verlag LNCS 3779), Springer, Vol. 3779, pp.2–13.

Friese, T., Smith, M. and Freisleben, B. (2006) 'GDT: A toolkit for grid service development', *Proceedings of the 3rd International Conference on Grid Service Engineering and Management*, pp.131–148.

Graham, S., Karmarkar, A., Mischkinsky, J., Robinson, I., Sedukhin, I., Treadwell, J., Srinivasan, L., *et al.* (2006) 'Web Services Resource Framework (WSRF)', http://www .oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

Heinzl, S., Mathes, M., Friese, T., Smith, M. and Freisleben, B. (2006) 'Flex-SwA: flexible exchange of binary data based on SOAP messages with attachments', *Proc. of the IEEE International Conference on Web Services*, Chicago: IEEE Press, pp.3–10.

Heinzl, S., Mathes, M. and Freisleben, B. (2008a) 'A web service communication policy for describing non-standard application requirements', *Proc. of the IEEE/IPSJ Symposium on Applications and the Internet (Saint 2008)*, IEEE Computer Society Press, pp.40–47.

Heinzl, S., Mathes, M. and Freisleben, B. (2008b) 'The grid browser: improving usability in service-oriented grids by automatically generating clients and handling data transfers', *Proceedings of the Fourth IEEE International Conference on eScience*, IEEE Press, pp.269–276.

Intel Corporation (2008) 'OpenCV', http://www.intel.com/technology/computing/opencv/.

Kajko-Mattsson, M., Lewis, G.A. and Smith, D.B. (2007) 'A framework for roles for development, evolution and maintenance of SOA-Based systems', *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, IEEE Computer Society, p.7.

Karastoyanova, D., Houspanossian, A., Cilia, M., Leymann, F. and Buchmann, A. (2005) 'Extending BPEL for run time adaptability', *EDOC '05: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, IEEE Computer Society, pp.15–26.

Kropp, A., Leue, C., Thompson, R., Braun, C., Broberg, J., Cassidy, M., Freedman, M., *et al.* (2003) 'Web services for remote portlets specification', *OASIS Standard*, August, http://www .oasis-open.org/committees/wsrp.

Lizcano, D., Soriano, J., Reyes, M. and Hierro, J.J. (2008) 'EzWeb/FAST: reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming web of services', *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services.*

Ma, R.Y., Wu, Y.W., Meng, X.X., Liu, S.J. and Pan, L. (2008) 'Grid-enabled workflow management system based on BPEL', *International Journal of High Performance Computing Applications*, Vol. 22, No. 3, pp.238–249.

Mozilla (2009a) 'Mozilla Application Framework', http://developer.mozilla.org/en/docs/Mozilla _Application_Framework_in_Detail.

Mozilla (2009b) 'XULRunner', http://developer.mozilla.org/en/docs/XULRunner.

Nambia, U., Ludaescher, B., Lin, K. and Baru, C. (2006) 'The GEON portal: accelerating knowledge discovery in the geosciences', *8th ACM International Workshop on Web Information and Data Management (WIDM 2006)*, ACM, pp.83–90.

Nestler, T. (2008) 'Towards a mashup-driven end-user programming of SOA-based applications', *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services.*

Nesvadba, J., Fonseca, P., Sinitsyn, A., de Lange, F., Thijssen, M., van Kaam, P., Liu, H., *et al.* (2005) 'Real-time and distributed AV content analysis system for consumer electronics networks', *Proc. of International Conference on Multimedia and Expo*, IEEE Computer Society, pp.1549–1552.

Olabarriaga, S., De boer, P., Maheshwari, K., Belloum, A., Snel, J., Nederveen, A. and Bouwhuis, M. (2006) 'Virtual lab for fMRI: bridging the usability gap', *Second IEEE International Conference on e-Science and Grid Computing (e-Science 2006).*

Salembier, P. (2002) 'Overview of the MPEG-7 standard and of future challenges for visual information analysis', *EURASIP J. Appl. Signal Process*, Vol. 2002, No. 1, pp.343–353.

Schulzrinne, H., Casner, S., Frederick, R. and Jacobson, V. (2003) 'RTP: a transport protocol for real-time applications', http://www.ietf.org/rfc/rfc3550.txt.

Seiler, D., Heinzl, S., Juhnke, E., Ewerth, R., Grauer, M. and Freisleben, B. (2008) 'Efficient data transmission in service workflows for distributed video content analysis', *Proceedings of the 6th International Conference on Advances in Mobile Computing & Multimedia*, ACM Press/OCG Book Series, pp.7–14.

Tanaka, K., Uehara, M. and Mori, H. (2009) 'The performance evaluation of a grid using Windows PCs', *Int. J. Web and Grid Services*, Vol. 4, No. 4, pp.395–417.

The GridSphere Project (2009) 'GridSphere portal framework', http://www.gridsphere.org/.

Viola, P. and Jones, M.J. (2004) 'Robust real-time face detection', *Int. J. Comput. Vision*, Vol. 57, No. 2, pp.137–154.

Zeng, H. and Miao, H. (2006) 'Data communication model of grid workflow', *ICEBE '06: Proceedings of the IEEE International Conference on e-Business Engineering*, Washington, DC: IEEE Computer Society, pp.647–654.

Notes

1 1.7 GB of memory, 1 EC2 Compute Unit (ECU) (1 virtual core with 1 ECU), 160 GB of instance storage, 32-bit platform. One ECU provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor.

2 1.7 GB of memory, 5 ECUs (2 virtual cores with 2.5 ECUs each), 350 GB of instance storage, 32-bit platform.

3 http://taverna.sourceforge.net/

Website

www.youtube.com