A Workflow Management Platform for Media Analysis in BPEL-based Grid Environments

Benjamin Ihle [1], Sebastian Kirch [1], Ernst Juhnke [2], Tim Dörnemann [2], Dominik Seiler [2], Bernd Freisleben [2]

[1] Fraunhofer IAIS, NetMedia, Schloss Birlinghoven, D-53757 Sankt Augustin, Germany
[2] Dept. of Mathematics and Computer Science, University of Marburg, Hans-Meerwein-Str. 3, D-35032 Marburg, Germany

ABSTRACT

Motivation: In many media applications, there is a need to index very large amounts of audiovisual content in a minimum of time. Due to their scalability, Grid infrastructures are well suited for processing a large number of media files. Within the BMBF[1]-funded research project *MediaGrid*[2], a distributed media analysis platform is currently being developed. It enables its users to analyze and index large quantities of multimedia data within a short amount of time. Grid services as distributed processing units and *BPEL4Grid* [10] as a scientific BPEL[3]-based workflow system are the foundations of the platform. To reduce the interface complexity and make the platform user-friendly, additional components have been added. This paper describes the architecture of the media analysis platform and the components involved.

Results: An architecture offering a generic job interface in combination with an extensible BPEL repository is presented. The complexity of the system is hidden from the user and shifted to the platform provider. Thereby, the barrier to use the system as an end user is lowered to a minimum.

Availability: Parts of the components that are combined to form the media analysis platform already exist (for instance, the generic job interface and the BPEL engine). Further enhancements such as the *BPEL Repository*, the *BPEL Adapter Repository* and the *Mapping and Invocation Component* will be integrated into the platform and will be made available in 2011.

Contact: benjamin.ihle@iais.fraunhofer.de

1 INTRODUCTION

The *MediaGrid* research project deals with distributed media analysis in Grid environments. The aim of the project is to build a Grid infrastructure that provides distributed services (encapsulating adequate algorithms) for analyzing audio and video content.

In *MediaGrid*, media analysis is conducted based on several processing steps (implemented as distributed Grid services) that are orchestrated using workflow technology. The services can be arranged differently in terms of their order of processing. To manage various sequences of distributed Grid services, *MediaGrid* uses the Business Process Execution Language (BPEL) – the de facto standard for business process modeling in today's enterprises. Moreover, BPEL is already widely used for the implementation and execution of workflows in scientific applications [1, 7].

This paper presents a system architecture for performing media analysis via a BPEL-based service platform in Grid environments. As a key feature, it shifts the complexity of a workflow system for distributed job processing from the customer to the platform provider. At the same time, the entire functionality provided by the Grid service remains available for the customer to the full extent. This is achieved by offering a generic interface that simplifies the usage of distributed services and significantly lowers the barrier to entry for the user to integrate the provided services in his or her own workflow. The paper gives an overview of the software components that are part of the workflow management system for service-oriented and distributed media applications.

This paper is organized as follows. Section 2 provides a general overview of the involved components and briefly describes their interaction. In Section 3, the BPEL4Grid engine used to execute Grid-enabled workflows is presented. Section 4 describes the components responsible for workflow management, while Section 5 depicts the user interface for job submission and monitoring. Projects related to the work presented in this paper are described in Section 6. Finally, Section 7 concludes the paper and outlines areas for future research.

2 ARCHITECTURAL OVERVIEW

In this section, the architecture of the system is described from a bird's-eye view to illustrate the interplay of the different components. The architecture includes several components, as depicted in Figure 1. The system's foundation is a service-oriented architecture based on Web and Grid services orchestrated using BPEL workflows. Since BPEL has a number of drawbacks with respect to WSRF-based Grid services, the Grid-enabled workflow modeling tool *Domain-Adaptable, Visual BPEL4WS Orchestrator* (DAVO) [5] and the *BPEL4Grid Engine* [10] are used to model and execute Grid-specific workflows.

The *BPEL4Grid Engine* only provides a workflow's interface description (WSDL document), but no additional metadata like information on parameters. This data is required to fulfill the main goal of this work: ease the invocation of the deployed workflows. Therefore, we introduce an additional component for

[1] Bundesministerium für Bildung und Forschung (German Federal Ministry of Education and Research)

[2] http://www.mediagrid-community.de

[3] Business Process Execution Language

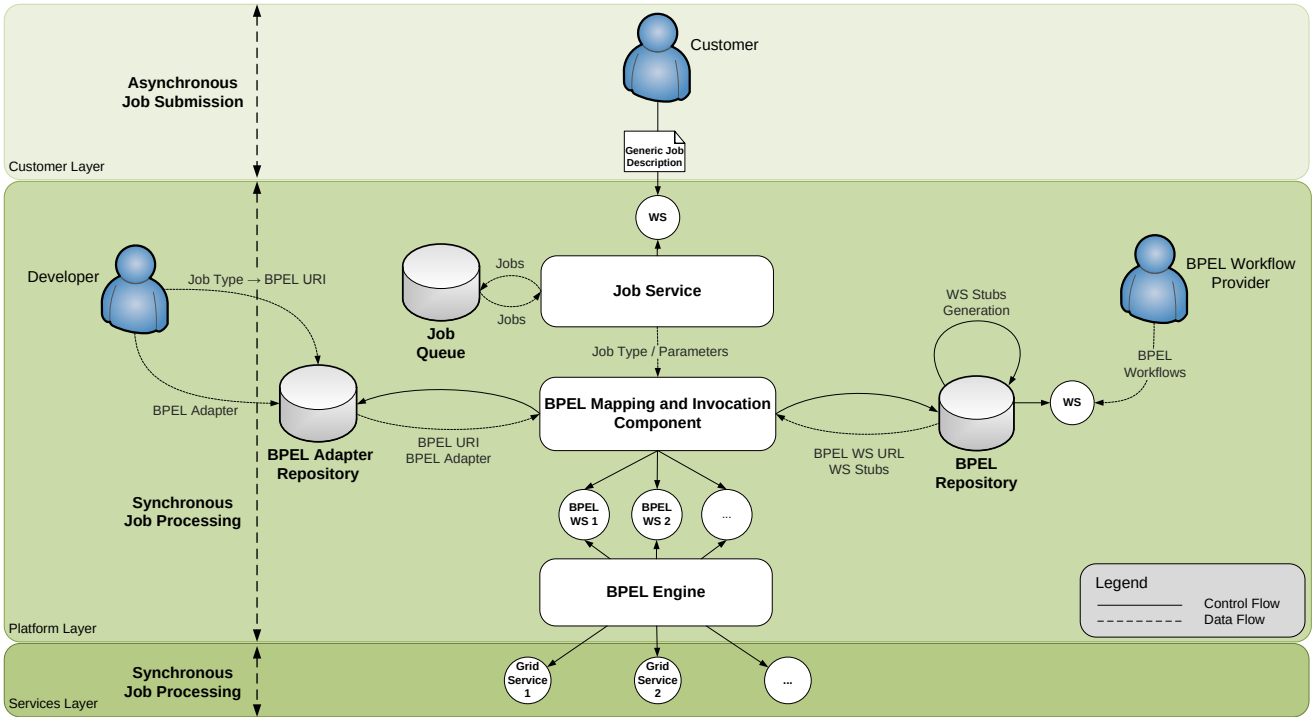

Fig. 1. Architecture overview of the proposed system

workflow registration: the *BPEL Repository*. Besides providing documentation on available workflows, it is also responsible for creating and storing Web Service stubs used to invoke the deployed workflows.

To invoke a workflow using a stub, an adapter object has to be provided that is capable of mapping the parameters submitted by the user to the generated stub. The *BPEL Adapter Repository* holds such an adapter for every available workflow.

The actual mapping and workflow invocation is performed by the *BPEL Mapping and Invocation Component* (BMIC). Every request to start the processing of a workflow is internally managed as a separate job object that contains information on the workflow to be invoked and its parameters. These job objects are organized in a *Job Queue*. When a new job is selected for processing from the *Job Queue*, the BMIC assembles all data objects necessary to invoke the workflow. Finally, it triggers the processing of the workflow.

Despite the variety of available workflows, there is only a single Web Service interface through which workflows can be started, managed and monitored: the *Job Service*. The *Job Service* interface defines a data type for submitting a *Generic Job Description*. It contains all the information needed to invoke a BPEL workflow such as the *Job Type* and *Parameters*. Users can query the *Job Service* for available workflows, choose a workflow they want to utilize and compose a *Generic Job Description* before it is submitted. Furthermore, the *Job Service* provides operations for monitoring a job's execution status and manage the execution lifecycle, i.e. to pause, resume or abort a running job.

3 GRID BPEL ENGINE

BPEL4WS has emerged from the earlier proposed XLANG[4] and Web Service Flow Language (WSFL)[5]. It enables the construction of complex web services composed of other web services that act as the basic activities in the process model of the newly constructed service. Access to the process is exposed by a execution engine through a web service interface, allowing those processes to be accessed by web service clients or to act as basic activities in other processes. BPEL features several basic activities to interact with the services being arranged in the workflow. These activities cover service interaction, data manipulation and the manipulation of the control flow. Structured activities, ranging from a sequential ordering of activities up to building directed acyclic graphs, allow the composition of complex operations. Furthermore, BPEL includes the features of scoping activities and specifying fault handlers and compensation handlers for scopes.

To communicate with stateless and stateful services that are built on top of the Web Services Resource Framework (WSRF)[6], an extended version of a BPEL engine [4] is used that facilitates the seamless integration of Grid applications in multimedia applications and vice versa. This extension maps the factory pattern of Grid/WSRF services into easy-to-use activities on the the BPEL side. Thus, the life-cycle management of a Grid service is reflected by the activities *GridCreateResourceInvoke* (GCRI), *GridDestroyResourceInvoke* (GDRI) while the invocation of a Grid service is performed by a *GridInvoke* (GI).

[4] http://msdn.microsoft.com/en-us/library/aa577463(v=bts.70).aspx

[5] http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[6] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

The implementation of our extensions to the BPEL standard is based on the BPEL engine developed by Active Endpoints[7], because the engine is quite robust and the source code is publicly available. Of special interest for our work are the Process Creation and Management and the Process component itself. The most important extension is the construct of *PartnerLinkSets* that encapsulates the handling of WSRF resources. A SOAP handler component has been developed that automatically inserts the Resource Key and other information needed to identify the resources into the SOAP Header of service calls. It is plugged into Apache Axis using the standard mechanism. Besides implementing classes for handling and storing properties of GI, GCRI and GDRI, the management GUI (web based) has been extended to reflect our changes to BPEL.

The GCRI activity identifies the corresponding partner link set before determining the resource link from it and constructing an invoke object. The created invoke object is added to the execution queue of the engine. As soon as the invoke is dequeued, any resource link information contained in that invoke is identified and the concrete endpoint is set in the Axis call. As soon as the response arrives, it is parsed and the resource key as well as the endpoint address are stored as a resource in the resource link. This information is handled by the partner link set data structure. When GI is called, it performs a lookup for the partner link set and identifies the resource link corresponding to it. The engine's natural strategy to resolve a partner link is to look them up by their names. To use this mechanism, we decided to introduce unique identifiers for resources that are stored in the invoke object. Hence, the engine can resolve the resource link during the creation of an Axis call object. Subsequently, the correct endpoint information is saved within the call and the information about the resource key is put into the message context. GDRI is used when a WSRF resource is not required anymore and therefore its lifetime should end. It constructs an invoke object in the same way GI does (but with the intention of destroying the resource).

Another obstacle BPEL workflows have to face are security constraints: BPEL has a number of drawbacks with respect to the more complex world of WSRF-based Grid computing. The BPEL security concept is not equipped to deal with complex multi-protocol Grid environments and does not integrate with the Grid Security Infrastructure (GSI). While BPEL is mainly focused on anonymous HTTPS-based TLS security or manual role-based authentication encoded in SOAP headers, Grid computing has a mandatory user-centric security approach using X.509 certificates that far exceeds the scope and capability of the BPEL security model. The Grid extensions mentioned above also feature the integration of GSI into the Grid service invocation that can also manage the lifetime of proxy certificates [6].

Due to the fact that BPEL is an XML-based web service composition language, the definition of a workflow is quite laborious and time-consuming without tool support. As a consequence, several research and commercial activities concentrate on the design and development of graphical BPEL workflow editors focusing on a clear visualization and syntactically correct mapping of the graphical representation to BPEL4WS code. Thus, existing BPEL workflow editors are only suitable for web service experts who are familiar with the details of web services and BPEL4WS. Non web service experts are normally overburdened by these editors.

The *Domain-Adaptable, Visual BPEL4WS Orchestrator* (DAVO) [5] is suitable for non web service experts. 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 which permit customization to specific domain needs. This increases usability for non web service experts.

The architecture of DAVO is based on a model-driven approach and follows the model-view-controller (MVC) design pattern [8]. Every element of the process is presented to the user through a view component with a corresponding controller, allowing editing operations. By visually adding elements to a workflow, changing properties and so on, the controller object corresponding to the action performed makes changes to the internal data model. Vice versa, changes to the internal data model trigger controller objects to update the visualization.

The mapping from the internal data model to executable BPEL code is performed by a code generation component. It generates at least three files from the internal model: a *bpel* file that contains the logic of the workflow, a *WSDL* file with the workflows interface description and a (non-standardized) deployment descriptor that contains runtime information such as service endpoints.

4 BPEL REPOSITORY AND WORKFLOW MANAGEMENT

4.1 BPEL Repository

When BPEL processes are deployed to a BPEL engine, interfaces are needed to handle workflow discovery and execution. The BPEL4Grid engine, which in turn is based on *ActiveBPEL*[8], is used in the *MediaGrid* project; it provides such a simple web service interface that can be used to discover available workflows, monitor their execution and manage the lifecycle of running processes. However, this simple interface lacks the possibility to obtain additional information to a process such as process semantics or a documentation of the process parameters. Furthermore, there is no single interface through which workflows that are hosted in remote BPEL engines can be discovered. What is needed is a centralized workflow repository capable of providing information about locally and remotely deployed processes to manage and discover available workflows at a single location.

Due to these requirements, a separate workflow registry component is used: the *BPEL Repository* (BR). Acting as a registry for BPEL processes, the BR stores information about all available processes. Similar to the management interface of the BPEL4Grid engine, processes can be added, altered or removed via its own management web service interface. It can also be used for process deployment, such that additional documentation about the process and its parameters can be provided along with the deployment.

Besides the workflows that run in the local BPEL engine, external workflow providers can use the interface to offer remotely hosted BPEL workflows to the user. All they need to provide is a WSDL interface description of the process. Additionally, they can add

[7] http://www.activevos.com/

[8] http://www.activevos.com/open-source.php

documentation about the process and its parameters that can be presented to the user when he or she intends to use the workflow.

Based on the WSDL interface description of a BPEL process, the BR is able to automatically generate web service stubs for the corresponding interface. These stubs are administered by the BR and are used by the *BPEL Mapping and Invocation Component* described in Subsection 4.3 to invoke the BPEL process.

4.2 BPEL Adapter Repository and Mapping Information

The *BPEL Adapter Repository* (BAR) stores the necessary objects for mapping the *Job Type* of a *Generic Job Description* to a specific BPEL process. For every *Job Type*, there exists a corresponding BPEL process that can be invoked by its *BPEL WS*[9] and that is accessible via the *BPEL WS URL*. As shown in Figure 1, the BAR returns an URI that specifies the location of the corresponding *BPEL WS URL* within the BR. To perform the mapping between the parameters specified by the *Generic Job Description* and the input variables of a certain *BPEL WS*, the repository additionally contains a *BPEL Adapter* (BA) for each *BPEL WS*, which contains all required mapping information. Thus, for every *Job Type* specified by the customer, the BAR provides an URI and a BA. Thereby, the repository only stores the mapping information but does not do any mapping at all. The mapping is actually done by the *BPEL Mapping and Invocation Component* that is described in Section 4.3.

The information about how to map a *Generic Job Description* to a specific *BPEL WS* including its input parameters has to be created manually. Since the mapping information can neither be extracted automatically from a *Generic Job Description* nor from a *BPEL WS*, it has to be implemented by a developer who knows both workflow descriptions and is aware of how the parameters have to be assigned to each other. Figure 1 shows the role of the developer. He or she is responsible for implementing the mapping of *Job Type*s to their corresponding *BPEL URI*s and BAs. To lower the barrier to entry for the customer, this relatively expensive and time consuming task is concealed from him or her. Thus, the effort to integrate the functionality of the platform in the customer's environment is reduced to a minimum.

4.3 BPEL Mapping and Invocation Component

The *BPEL Mapping and Invocation Component* (BMIC) performs the mapping between the data of the *Generic Job Description* provided by the customer and the associated *BPEL WS*. Moreover, it invokes the corresponding process. To know how to map the parameters to the input variables of the specific *BPEL WS*, a retrieval of various data objects is necessary beforehand. Thereby, the user's input data in the form of a *Generic Job Description* and the WSDL URL of the corresponding *BPEL WS* as well as an appropriate BA are used. These objects are retrieved from internal repositories. The entire process of data mapping is described in more detail below. Figure 2 presents a graphical overview of the mapping process showing its single steps in numerical order.

A new job processing is triggered by the customer by providing a new *Generic Job Description*. It comprises a *Job Type* that corresponds to a certain workflow type and a number of parameters describing necessary input values. First (1), the BMIC needs to know which specific BPEL process belongs to the *Job Type* provided by the customer. Therefore, it puts a request to the BAR specifying the *Job Type* that the user has chosen. The BAR matches the provided *Job Type* to an URI pointing to the location of the *BPEL WS* to call (2) and returns this URI along with the corresponding BA (3). To assemble a web service client to invoke functions of the destined *BPEL WS*, a request is put to the BR providing the URI of the mapped BPEL workflow (4). The BR does the matching for the provided URI (5) and returns the *BPEL WS URL* that points to the assigned *BPEL WS* (6). Apart from the *BPEL WS URL*, the BR provides *WS Stubs* generated from the WSDL description of the *BPEL WS URL*. Now that all necessary information is available to create an user-specific request to the corresponding *BPEL WS*, the actual mapping process takes place (7). The parameters originally provided by the customer through the *Generic Job Description* are mapped to the input variables of the *WS Stubs* of the assigned *BPEL WS*. The result of the mapping process is a parametrized web service client. It is used to invoke the *BPEL WS* that has been identified for the user-chosen *Job Type*. Finally, the corresponding *BPEL WS* is invoked (8) and the processing of the workflow steps begins.

5 GENERIC JOB INTERFACE

BPEL processes mostly represent a chain of sequential and parallel processing steps and are invoked using web service interfaces. Each process provides such an interface specifying the operations available and the parameters needed to invoke the operation[10]. Since BPEL processes can differ significantly depending on the task they perform, the corresponding web service interfaces can also differ. A user would have to implement a specific client for every new process he or she may want to use. A drawback of this approach is that a growing number of different processes increases the complexity and the effort to integrate them into an application.

The main goal of the *Job Service* interface is to bypass this shortcoming of a service-oriented architecture by providing a single, generic web service interface for workflow invocation. This interface acts as a façade to hide the workflow invocation logic from the user by transferring the complexity towards the platform provider. Instead of choosing a workflow and generating stubs to invoke this workflow, the user composes a *Generic Job Description* which is interpreted by the *Job Service* as shown in Figure 1. The job description contains a specific *Job Type* and the *Parameters* necessary for its execution. An extraction of the XML schema code representing the job description is shown in Listing 1.

Every *Job Type* uniquely maps to a corresponding *BPEL WS* able to perform the desired task. By using the mapping component described in Subsection 4.3, the parameters provided are matched to the corresponding BPEL parameters. Since stub classes are created automatically by the *BPEL Repository*, applications using the platform only have to implement a single web service interface.

The *Job Service* interface cannot only be used for job submission. It also provides all the information necessary to compose a job description. Applications can query the *Job Service* for available *Job Types* and present them to the user. The customer can easily choose a *Job Type* from a number of already existing *Job Type*s

[9] Web Service

[10] For simplification, we assume that a workflow has only one operation. "Invoking a workflow" means invoking this single operation.

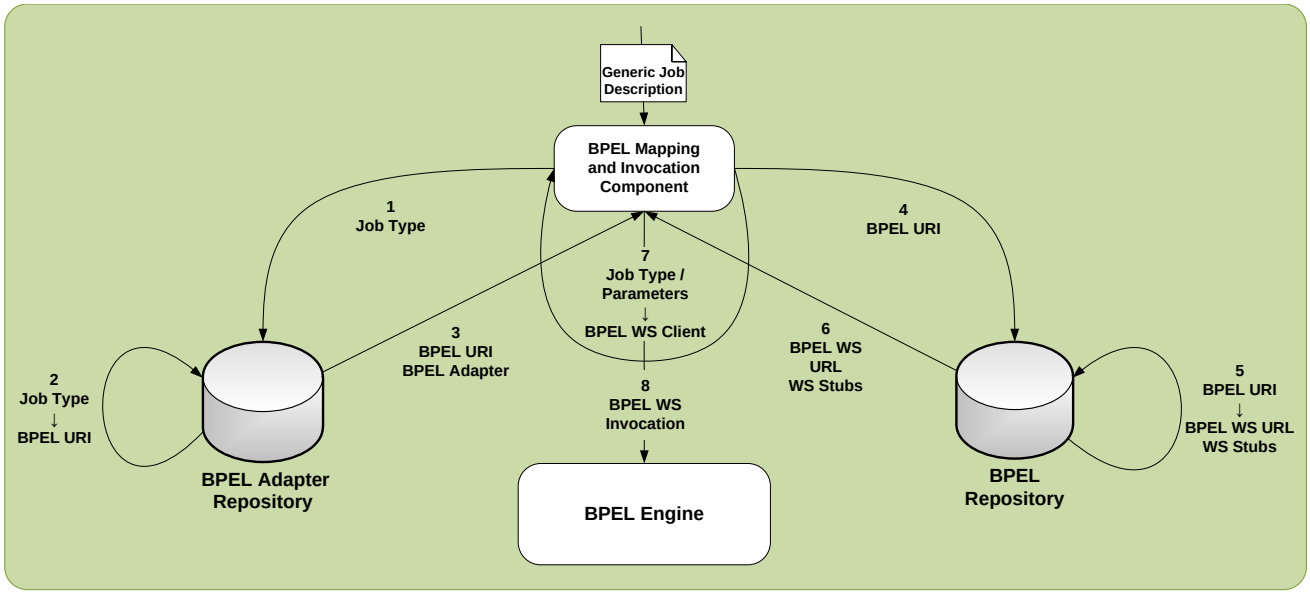

Fig. 2. Detailed view on the single steps of a mapping process

and has to provide required parameters. After a job description has been composed, it is passed to the *Job Service* that in turn starts the workflow processing. The *Job Service* responds with a unique job ID which can be used to manage and monitor the job execution process. Besides the ability to add new jobs, the *Job Service* interface provides operations to query a job's state and to *abort*, *suspend* or *resume* a running job using the job ID.

```
<complexType name="jobDescription">
  <sequence>
    <element name="jobType" type="tns:jobType"/>
    <element name="title" type="string"/>
  </sequence>
</complexType>

<complexType name="jobType">
  <sequence>
    <element name="comment" type="string"/>
    <element name="parameters" type="tns:jobParameter"
        minOccurs="0" maxOccurs="unbounded"/>
    <element name="type" type="string"/>
  </sequence>
</complexType>

<complexType name="jobParameter">
  <sequence>
    <element name="comment" type="string"/>
    <element name="name" type="string"/>
    <element name="type" type="QName"/>
    <element name="value" type="base64Binary"/>
  </sequence>
</complexType>
```

Listing 1. XML schema code representing the Job Description

After the *Job Service* has received a job description, a new job is created and added to the *Job Queue* from where it is further processed. Hence, the *Job Service* decouples the asynchronous submission of jobs from the synchronous job execution. Jobs can be submitted independently of job execution. With the job ID, users can determine a job's state and manage its execution lifecycle. Since the *BPEL Engine* is in control of process execution, operations like *suspend*, *resume* or *abort* are forwarded to the engine's management interface.

More complex information, e.g. about the implementation of certain BPEL processes, remains hidden from the user. Thus, the effort to integrate the functionality of the platform in a user's own environment is reduced to a minimum.

6 RELATED WORK

The CASSANDRA framework [11] is a distributed multimedia content analysis system that is based on a service-oriented architecture and aims to facilitate the composition of applications from distributed components on a network of cooperating devices (like PC 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 that is synchronized by a master repository. Service composition is initiated by a special coordination component. Each Service Unit has a control and a data streaming interface. The control interface is based on UPnP, while the streaming interface is based on TCP/IP. This framework does not use web services to build a SOA, neither for the service definition nor for workflow composition; it uses UPnP. Since service composition is performed by a special component, the above mentioned 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. 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.

Blower et al. [3] have 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 file-sharing protocol Styx and allows data streaming from service to service over transport

protocols like TCP/IP or UDP. A SGS can be used in a web service or WSRF environment. An orchestration into workflows is also possible through simple shell scripts or graphical workflow system like the Taverna workbench[11]. In contrast to SGS, our architecture is based on the de-facto standard for business workflows, namely BPEL, and can thus be integrated into business processes. The granularity of the SGS is very coarse, since a SGS can only wrap whole executables. Our approach works on the web or Grid service level, i.e., on methods or functions.

With a large scale content analysis engine, Gibbon et al. [9] have built a distributed system to assist content-based video retrieval and related applications. The system consists of multiple specialized servers with a centralized database. Video data is ingested through acquisition servers and delegated to certain processing servers. The system relies on the MPEG-7 standard to store and distribute metadata. It makes use of web services to expose processing operations, but the task of data transport is delegated to a subsystem. Thus, the data transfer is performed outside of the web service scope. Furthermore, the system does not rely on standardized workflow systems.

An architecture and an API to support a distributed data flow model in conjunction with a centralized control flow is presented by Barker et al. [2]. To provide a non-centralized data flow from one service to another service, the authors make use of proxies that are responsible for data management and service invocation. The proxies are placed near by the service that they should administrate. A service invocation has to be passed over a proxy, the produced data is saved through the proxy and a reference to this data is delegated back to the calling centralized workflow engine. This solution is just focused on (stateless) web services and does not directly support WSRF-based services that 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 local disk, a resource competition between service and proxies may occur.

7 CONCLUSION

In this paper, we have presented an architecture for a generic Grid workflow management platform applied to media analysis. By specifying a single *Job Service* interface, the barrier to entry for using the workflows is lowered to a minimum without losing any functionality. No matter how many different workflows the customer likes to process, just one interface has to be implemented. The complexity of workflow processing is transferred from the customer to the platform provider. Acting as a façade for workflow processing, the *Job Service* aggregates all functions needed to invoke, manage and monitor workflows. Furthermore, external workflows can be added to the *BPEL Repository* to support the integration of workflows operated by third party providers.

The presented solution is currently under development and is estimated to be available as a prototype in 2011. Areas for future work are: a) completing the implementation, b) using the architecture for various analysis applications, and c) evaluating the performance and scalability of the proposed approach.

[11] http://taverna.sourceforge.net/

REFERENCES

[1]A. Akram, D. Meredith, and R. Allan. Evaluation of BPEL to Scientific Workflows. In *Proc. of the Sixth IEEE Int. Symposium on Cluster Computing and the Grid*, pages 269–274. IEEE, 2006.

[2]A. Barker, J. B. Weissman, and J. van Hemert. Orchestrating Data-Centric Workflows. In *CCGRID*, pages 210–217, 2008.

[3]J. D. Blower, A. B. Harrison, and K. Haines. Styx Grid Services: Lightweight, Easy-to-Use Middleware for Scientific Workflows. In *International Conference on Computational Science (3)*, pages 996–1003, 2006.

[4]T. Dörnemann, T. Friese, S. Herdt, E. Juhnke, and B. Freisleben. Grid Workflow Modelling Using Grid-Specific BPEL Extensions. In *Proceedings of German e-Science Conference (GES)*, 2007.

[5]T. Dörnemann, M. Mathes, R. Schwarzkopf, E. Juhnke, and B. Freisleben. DAVO: A Domain-Adaptable, Visual BPEL4WS Orchestrator. In *Proceedings of the 23^{rd} IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 121–128. IEEE Computer Society Press, 2009.

[6]T. Dörnemann, M. Smith, and B. Freisleben. Composition and Execution of Secure Workflows in WSRF-Grids. In *Proceedings of the 8^{th} IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 122–129. IEEE Computer Society Press, 2008.

[7]W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S. Price. Grid Service Orchestration using the Business Process Execution Language. In *Journal of Grid Computing*, volume 3, pages 283–304. Springer, 2005.

[8]E. Gamma, R. Helm, and R. E. Johnson. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[9]D. Gibbon and Z. Liu. Large scale content analysis engine. In *Proc. of the First ACM workshop on Large-scale multimedia retrieval and mining*, pages 97–104. ACM, 2009.

[10]E. Juhnke, T. Dörnemann, R. Schwarzkopf, and B. Freisleben. Security, Fault Tolerance and Modeling of Grid Workflows in BPEL4Grid. In *Proceedings of Software Engineering 2010, Grid Workflow Workshop (GWW-10)*, pages 193–200. , 2010.

[11]J. Nesvadba, P. Fonseca, A. Sinitsyn, F. de Lange, M. Thijssen, P. van Kaam, H. Liu, R. van Leeuwen, J. Lukkien, A. Korostelev, J. Ypma, B. Kroon, H. Celik, A. Hanjalic, U. Naci, J. Benois-Pineau, P. de With, and J. Han. Real-Time and Distributed AV Content Analysis System for Consumer Electronics Networks. In *Proc. of International Conference on Multimedia and Expo*, pages 1549–1552. IEEE Computer Society, 2005.