An Execution Environment for Mathematical Services based on WSRF and WS-BPEL *

Andreas Duscher
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, Linz, Austria
andreas.duscher@risc.uni-linz.ac.at

December 2005

Abstract

This paper describes the architecture of an execution environment for mathematical services using current web service technologies. This environment allows to execute manually created process descriptions that specify the dynamic behavior of stateful mathematical services; the individual interaction steps needed for obtaining a result are hidden by the environment. Building on previous work done in this area, an approach for transforming descriptions in Mathematical Services Description Language (MSDL) is introduced.

Contents

*This work was sponsored by the FWF (Austrian Science Fund) Project P17643-N04 "MathBroker II: Brokering Distributed Mathematical Services"

1 Introduction

Mathematical services are web services that provide solutions to mathematical problems. Due to the nature of mathematics they usually operate in a semantically rich domain. Current web service technologies (e.g. WSDL[1] and SOAP[2]) mainly cover static aspects on a syntactic level. Projects like "MathBroker" or "Mathematics on the Net" described in Section 2 have thus extended web service technologies by means to encode semantic information about mathematical services.

Moreover web service interfaces are conceptually similar to remote procedure calls; interaction protocols that consist of multiple calls have to be written manually which is a tedious and error-prone task. This is a problem for mathematical services because of two reasons: First, mathematical services are usually backed by software that is intended for solving problems in a certain area of mathematics. Such software packages often need some initialization steps (e.g. for loading corresponding libraries) and/or termination steps (e.g. for freeing allocated resources). Second, mathematical services are involved in an intensive dialog with a client to produce a result, i.e. according to a certain interaction protocol a sequence of messages has to be exchanged between both parties.

This paper describes an execution framework for mathematical services based on *Web Services Resource Framework* (WSRF) and *Web Services Business Process Execution Language* (WS-BPEL). WSRF is a set of specifications that allow to model and access stateful resources in a standardized way. WS-BPEL specifies a notation for describing the behavior and the interaction of a process instance relative to its participating Web services. The execution environment described in this paper combines both standards to allow the dynamic and stateful interaction between a process instance and its involving resources that are accessed via stateful web services. The ultimate goal (that is not yet covered in this paper) is the automated derivation of such an interaction protocol from semantic service descriptions.

The structure of this paper is as follows: After a sketch of the related work in Section 2, Section 3 gives an overall example of a process instance and a stateful mathematical service deployed in the execution environment. Section 4 describes how to apply stateful communication patterns to mathematical services with WSRF. Section 5 presents a process-oriented approach to describe the behavior of mathematical services. Finally Section 7 concludes the paper and gives an overview of the future directions. The Appendix lists the documents describing mathematical services and processes for further inspection.
The software described in this paper can be evaluated at the demonstration site *http://dragonfly.risc.uni-linz.ac.at:8080/*.

2 Related Work

In the past two projects have dealt with brokering mathematical web services. In the "MathBroker" project [19] web service technologies have been applied to

build a basic infrastructure for web services that offer mathematical problem solving capabilities. The implemented samples use SOAP [2] to transfer mathematical objects in the OpenMath representation and WSDL [4] for describing the service interfaces. Two major project results build on these technologies: The Mathematical Service Description Language (MSDL) for describing mathematical services and a registry for storing and querying such descriptions. MSDL is formulated in XML and encodes the following concepts [23]: mathematical problems, algorithms solving these problems, implementations of the algorithms, machines as execution platforms, services described with WSDL documents that are located on these machines and realizations that link services to implementations. The registry bases on the Oasis ebXML [26] reference implementation and allows via a Java API to query and manipulate the MSDL descriptions.

In the European "Mathematics on the Net" (MONET) project [21] a prototype architecture for mathematical web services was developed which consists of clients and services, a broker for discovering services by clients [27] and a manager for handling object persistence. MONET was launched simultaneously with the "MathBroker" project and both influenced each others. While the MONET project has taken over the idea of Mathematical Service Description Language (MSDL) and expressed its own version of it, the "Mathbroker" project redefined the original MSDL as an extension of the new version created by MONET [28]. In the final stages of MONET, it was investigated how to encode the MONET language in the Web Ontology Language OWL [30] such that brokers for mathematical services can make use of reasoning tools of the Semantic Web community. While the OWL tools were found to be still experimental, this was considered as a promising direction for the future [22].

3 The Overall Picture

This section gives an overall picture of the general problem domain and the work described in this paper. It consists of two parts. The first part presents an introductory example addressing the tasks that may occur when communicating and interacting with mathematical services. The second part presents the general architecture of the execution environment, whose details are described in the remainder of the the paper.

Example Scenario Let us consider the following scenario: a user with a mathematical problem to be solved is aware of a certain mathematical service that can fulfill this request. For solving the problem with help of this service, the following sequence of interaction steps has to be executed by the service:

- Creating a temporary instance of the backing mathematical software package.
- Loading the specific libraries needed for providing a result.
- Receiving the request (including the input for the software package) to compute a result.

- Returning the answer containing the output the software package.
- Freeing any additionally allocated resources and terminating the created instance.

The mathematical software package is exposed through a Web service interface. So the user needs not directly interact with the software package over proprietary protocols, but can access it via web service technologies (e.g. SOAP). Although these technologies allow a standardized communication with a web service, they mainly address static and syntactic issues; as a consequence, the service provider has to offer additional documentation about the service's dynamic behavior. To overcome this deficiency the execution environment presented in this paper performs two important tasks.

- First it manages and stores process descriptions that specify the interaction steps and their correct execution order. These process descriptions represent the service's dynamic behavior.
- Second, it allows to create executable process instances that directly communicate with the mathematical service to obtain a result. From the user's perspective, it suffices to invoke the execution environment that creates a process instance and independently executes the mentioned interaction steps to return the result.

Execution Environment Figure 1 gives a general overview of the architecture described in the above example. Two main components can be identified, namely a mathematical service and an engine for process execution:

- The mathematical service called *GAPService* (see Section 4) exposes the functionality of the computer algebra system GAP. Requests made to this service are internally processed by the *GAPInstanceManager* which manages the life cycle of the GAP instances and redirects the requests to the right instance.
- The execution engine is based on Twister (see Section 5.2) and executes deployed process descriptions, that are declared in the WS-BPEL language. Three web services expose the execution engine's functionality:
  - TwisterEngineWS: This service allows to directly send a message to a deployed process to start the process execution.
  - TwisterEngineAdminWS: This service is used for administrative purposes (i.e. deleting deployed process and web service descriptions).
  - TwisterDeployerWS: Using this service new process and web service descriptions can be deployed.

  As mentioned above, the user need not communicate with the mathematical service. The request for solving a mathematical problem is sent directly to the execution engine and consists of two parts: the name of the

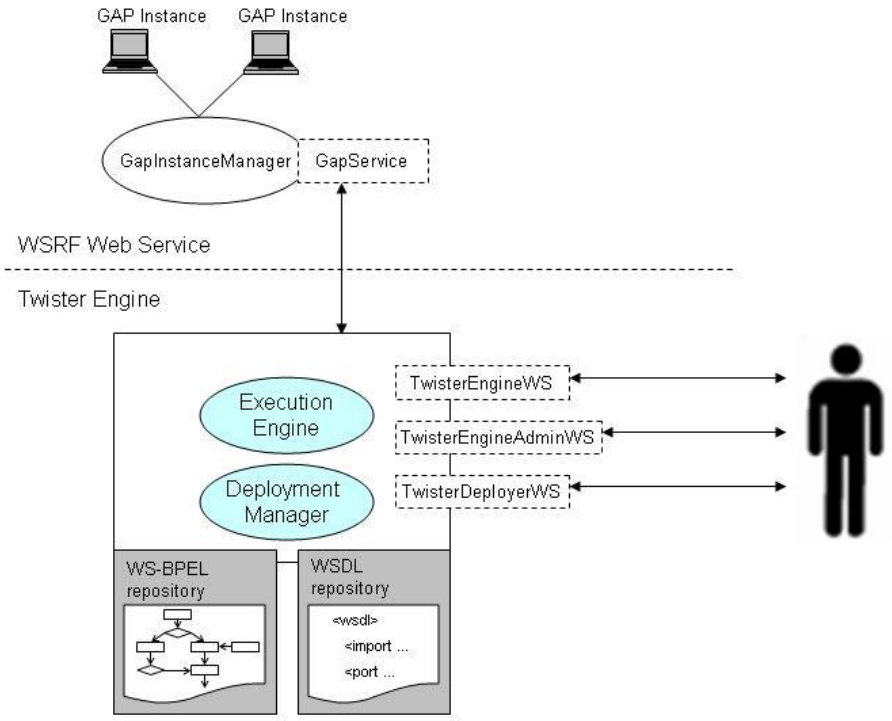

Figure 1: Architecture

process that can solve the problem and the initial data that is needed for finding a problem solution. After successful execution the engine returns the result to the user; the several interaction steps that are involved to find a problem solution are hidden from the user.

The next sections describe the involved standards and technologies of this architecture in more detail.

4 Mathematical Services in WSRF

The following subsections describe the specifications and mechanisms used to realize a sample mathematical service. Section 4.1 presents the WS-Addressing specification that allows to uniformly describe service endpoint references and corresponding resources. Section 4.2 presents an overview of WSRF specifications. Finally Section 4.3 introduces a mathematical service that is backed by the computer algebra system GAP [8].

4.1 WS-Addressing

The WS-Addressing specification [6] provides a transport and application neutral mechanism for addressing Web services and messages. It introduces two no concepts to SOAP: *endpoint references (EPR)* and *message information headers*.

A Web service endpoint is a reference to an entity or resource which is targeted by a Web service. Endpoint references carry the information to identify a Web service endpoint. They enable the identification and description of instances that are the result of stateful service interactions. Moreover they allow to define addresses for individual messages sent to and received from Web services. To deal with this last scenario the WS-Addressing specification defines a set of message information headers that facilitate the uniform addressing of messages that are decoupled from the underlying transport model. These message information headers carry additionally meta information about a message including addressing for source and destination endpoints as well as message identity.

4.1.1 EPR Information Model

In many cases messages are directly aimed at Web services and the needed addressing information concerning this message can be easily described within a URL. But in contrast, it might occur that messages are targeted to specific entities or resources that are tightly coupled to a Web service. For example, a mathematical service has to coordinate several different computation tasks. The service has to associate most of the incoming messages with a specific task instance that it manages and not with the mathematical service itself. The WS-Addressing specification provides a mechanism called an endpoint reference for targeting and addressing entities that are managed by a service. While such information could be encoded in an ad-hoc manner within the URL of the service [7], the mechanism of endpoint references provides a standardized XML fragment that allows a structured and more SOAP-conform approach. The information model for endpoint references reflects this structured approach. According to [6] it consists of the following elements:

Address An required element that carries the URI of an endpoint.

ReferenceProperties Reference properties identify an entity or resource targeted by a service endpoint. The properties are represented by child element and fully depend on the service publisher's schema definition. It is assumed that endpoint references with different reference properties may act on different messages or may have different meta data.

ReferenceParameters The difference between a reference parameter and a reference property is the intended usage. Reference parameters describe information interaction Both, the interpretation of reference properties and reference parameters may depend on the specified protocol binding.

Policy The policy specifies the requirements and capabilities of the endpoint.

Additional elements from EPR's information model are listed and explained in more detail in the specification document (see [6]).

Figure 1 shows an XML-based representation of a concrete endpoint reference. The "Address" element specifies the service endpoint. Inside the "ResourceProperties" the service publisher can place arbitrary elements that addresses the entity or resource instance.

```
<wsa:EndpointReference>
    <wsa:Address>
        http://localhost:8080/wsrf/services/GapPort
    </wsa:Address>
    <wsa:ReferenceProperties>
        <gap:ResourceIdentifier>1</gap:ResourceIdentifier>
    </wsa:ReferenceProperties>
</wsa:EndpointReference>
```

Figure 1: Example of an endpoint reference

When messages are sent to an endpoint, elements from endpoint references are mapped to the message according to a defined binding policy. The SOAP binding for endpoint references applies two major rules:

- URIs within an "Address" element are copied literally to a "To" element in the SOAP header.
- Each child element of "ReferenceProperties" and "ReferenceParameters" elements is copied to the SOAP header, maintaining the structure of the copied element.

According to the mentioned binding rules Figure 2 shows the construction of a SOAP message derived from the endpoint reference in Figure 1.

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   xmlns:wsa="..." xmlns:gap="...">
   <soap:Header>
        ...
      <wsa:To>
        http://localhost:8080/wsrf/services/GapPort
      </wsa:To>
      <gap:ResourceIdentifier>1</gap:ResourceIdentifier>
        ...
   </soap:Header>
   <soap:Body>
        ...
   </soap:Body>
</soap:Envelope>
```

Figure 2: Mapping from endpoint references to SOAP headers

The usage of reference properties and reference parameters is not clearly specified, although it exists a proposed usage scenario for each element. Both concepts can be mixed up, because both will be mapped to elements in a SOAP header. From a technical point of view no difference exists.

4.1.2 Message Information Headers

Message information headers enhance SOAP messages with new meta data about the web service endpoint. The following part describes the mostly used elements.

To An required element that carries the URI of an endpoint.

From Endpoint reference where the message originated from.

ReplyTo Endpoint reference that identifies the intended receiver for message replies.

MessageID A URI that uniquely defines the message.

4.2 The Web Services Resource Framework

The Web Services Resource Framework (WSRF)[3] consists of a set of specifications that allows modeling and accessing persistent resources through stateful Web services in a standardized way. Web services often imply the need for some form of persistence during communication sessions with a client. This is obvious in cases where the execution of one operation influences the succeeding operations. A Web service is characterized by sent and received messages which are defined in a WSDL document. Any resource that is manipulated by a Web service additionally needs to be unambiguously identified by the exchanged messages (see [4]). Not only that this type of interaction generates additional message overhead, but no uniform method for encoding the resource identifier existed so far.

WSRF introduces the idea of an XML schema definition, called *Resource Properties Document*, that represents the resource's properties. It is referenced in a WSDL description and explicitly describes the resource with which a client interacts. The following section shortly describes the WSRF's set of specifications.

4.2.1 WS-* Specifications

This section shortly describes the set of commonly used specifications. In general the specifications are under current development

WS-ResourceProperties (WSRF-RP) This specification standardizes the definition of properties of a resource as part of a Web service interface. It defines the relation between resources and Web services [10].

WS-ResourceLifetime (WSRF-RL) This specification defines messages for terminating and/or destroying resources and properties that allow to monitor a resource's lifetime information [11].

WS-ServiceGroup (WSRF-SG) This specification allows to group Web services and its resources together for a domain-specific purpose [12].

WS-BaseFaults (WSRF-BF) This specification allows to commonly define faults that may occur during execution [13]. It defines an XML Schema type for defining base faults along with rules on how the fault is used.

4.3 The GAPService

We have used the mechanisms described in the previous subsections, to implement a sample mathematical service called *GAPService*, that allows access to instances of the computer algebra system GAP [8]. It builds on Apache WSRF [9], which is a Java implementation of the WSRF family of specifications. Depending on the concrete mathematical problem to be solved, each client may have different requirements to the GAP system, thus a single client needs its own instance for the duration of interaction. The *GAPService* manages the creation and life cycle of every instance. Moreover it brokers client requests to matching instances. In WSRF notation a GAP instance would be called a *resource*.

A service which exploits WSRF must define a *resource properties document* and may then use a combination of standard and service-specific operation definitions to define the messages which interact with the document and the resource which it describes. Figure 3 presents a shortened version of this document.

```
<xsd:element name="GapResourceProperties">
 <xsd:complexType>
  <xsd:sequence>
     <xsd:element ref="gap-pr:Reference"/>
     <xsd:element ref="gap-pr:State"/>
     <xsd:element ref="gap-pr:LastAction"/>
     <xsd:element ref="wsrf-rl:CurrentTime"/>
     <xsd:element ref="wsrf-rl:TerminationTime"/>
      ...
  </xsd:sequence>
 </xsd:complexType>
 <xsd:element name="Reference"
              type="wsa:EndpointReferenceType"/>
 <xsd:element name="State" type="gap-pr:StateType"/>
 <xsd:simpleType name="StateType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="ready"/>
    <xsd:enumeration value="processing"/>
    <xsd:enumeration value="aborted"/>
    <xsd:enumeration value="completed"/>
```

```
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="LastAction" type="xsd:string"/>
  ...
</xsd:element>
```

Figure 3: Definition of resource properties in gap-properties.xsd

The description consists of a set of resource property elements that characterize a GAP instance. "State" is an enumerated type describing the resource's current processing state. "LastAction" contains the last processed GAP-specific command. As the namespace indicates "CurrentTime" and "TerminationTime" are predefined elements from the WSRF-RL specification [11] and provide information about timing-related issues. The resource property element "Reference" is of type endpoint reference. It allows relevant clients to obtain structured and standardized information about the Web service endpoint and the GAP instance (see Section 4.1.1). *GapResourceProperties* is the element's name that describes the resource, and its properties on a syntactical level. Figure 4 shows how to attach a resource definition to a specific port type.

```
<wsdl:definitions
    xmlns:wsrf-rp="..."
    xmlns:gap-pr=
  "http://risc.uni-linz.ac.at/services/gap/gap-properties.xsd"
    ...
    <wsdl:portType name="GapPortType"
      wsrf-rp:ResourceProperties="gap-pr:GapResourceProperties">

      <wsdl:operation ...
    ...
    </wsdl:portType>
</wsdl:definitions>
```

Figure 4: WSDL file and its link to resource properties

In this section the mechanisms for defining resources in a WSRF-conform manner and the relation to WSDL documents have been presented by means of GAP instances. In Section 4.2.1, a more detailed and general view on the construction of resource properties documents and its relation to WSDL is given.

4.3.1 The Creation of a Service Instance

After defining the resource properties that represent a GAP instance, this section describes the definition and usage of service-specific operation for creating such an instance. Although the WSRF-RP specification provides a set of predefined operations for manipulating and accessing resource properties, a WSRF web service is not restricted to this operations. New operations and message elements may be introduced.

Figure 5 shows another fragment of gap-properties.xsd that defines messages which are intended for the *GAPService*. These message elements reside

in the same document and share the same namespace as the resource property elements, which is for convenience and is not a restriction demanded by any WSRF specification.

```
<xsd:element name="CreateInstanceRequest"/>
<xsd:element name="CreateInstanceResponse">
  <xsd:complexType>
    <xsd:sequence>
     <xsd:element ref="gap-pr:Reference"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 5: Service-specific message elements in gap-properties.xsd

"CreateInstanceRequest" is an empty element which indicates the request for instance creation. In case of successful creation an element of type "CreateInstanceResponse" is returned as response which contains an endpoint reference that can be used for further execution steps and instance identification.

Figure 6 pictures the relation between schema-specific message elements (as defined in Figure 5), WSDL message elements and its corresponding WSDL operations. Although the WSDL specifaction permits several part elements, WSRF web services only facilitate one part in their WSDL message elements. So one uniquely defined schema-specific message element may be utilized to contain the appropriate information. This proceeding leads to a more message-oriented and implementation-independent approach, because no ambiguous SOAP encoding is used and eventually the web service endpoint is responsible for processing the message. The service-specific operation "createInstance" uses the WSDL message elements as input and output parameters, which are described by their corresponding schema-specific messages. The operation applies a synchronous communication pattern. Additionally a custom fault type is specified, which is in cases of errors enhanced with further information.

```
<wsdl:message name="CreateInstanceRequest">
  <wsdl:part name="CreateInstanceRequest"
    element="gap-pr:CreateInstanceRequest"/>
</wsdl:message>
<wsdl:message name="CreateInstanceResponse">
  <wsdl:part name="CreateInstanceResponse"
    element="gap-pr:CreateInstanceResponse"/>
</wsdl:message>
...
<wsdl:portType ...>
  ...
  <wsdl:operation name="createInstance">
    <wsdl:input name="CreateInstanceRequest"
      message="gap:CreateInstanceRequest"/>
    <wsdl:output name="CreateInstanceResponse"
      message="gap:CreateInstanceResponse"/>
    <wsdl:fault name="GeneralClientFault"
```

```
            message="gap:GeneralClientFault"/>
  </wsdl:operation>
  ...
</wsdl:portType>
```

Figure 6: Service-specific operation and messages in GapService.wsdl

The last part of this section deals with the construction of SOAP messages that can be processed by the *GAPService* to create an instance. Figure 7 shows a typical SOAP envelope consisting of a header and a body element. Information about the service endpoint is encoded in the header applying the concepts of message information headers (more in Section 4.1.2). The body element contains one message element that indicates the request for instance creation. Due to the defined relation between the schema-specific message element "CreateInstanceRequest" and the WSDL operation "createInstance" in GapService.wsdl (see Appendix 8.2), the request is brokered to the matching method.

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:gap="http://risc.uni_linz.ac.at/services/gap"
  xmlns:gap-pr=
 "http://risc.uni-linz.ac.at/services/gap/gap-properties.xsd">

  <Header xmlns:wsa=
   "http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <wsa:To mustUnderstand="1">
        http://localhost:8080/wsrf/services/GapPort
    </wsa:To>
    ...
  </Header>
  <Body>
   <gap-pr:CreateInstanceRequest/>
  </Body>
</Envelope>
```

Figure 7: Creation request message

In the actual implementation this custom method creates a new GAP instance and assigns an unique resource identifier. A scheduler mechanism manages and controlls created instances and automatically terminates idle ones. After the successful initialization of a GAP instance the operation returns a message element of type "CreateInstanceResponse". It contains according to the schema definition an endpoint reference (see Figure 8), which clients may use for further instance identification.

```
<Envelope>
...
<Body>
  <gap-pr:CreateInstanceResponse>
   <wsa:Address>
       http://10.0.1.2:8080/wsrf/services/GapPort
   </wsa:Address>
```

```
    <wsa:ReferenceProperties>
      <gap:ResourceIdentifier>1</gap:ResourceIdentifier>
    </wsa:ReferenceProperties>
    <wsa:PortType>gap:GapPortType</wsa:PortType>
    <wsa:ServiceName PortName="GapPort"/>
  </gap-pr:CreateInstanceResponse>
</Body>
</Envelope>
```

Figure 8: Creation response message

4.3.2 The Interaction with the Instance

In Section 4.3.1 the definition and relation of schema-specific message elements, WSDL message elements and WSDL operations has been explained in detail. This section focuses on the interactions that take place after successful instance creation and general WSRF communication patterns.

To access resource properties, several predefined WSRF-specific message elements exist. The message element "GetResourcePropertyDocument" is brokered to an existing method that is provided by the WSRF framework. This method generates and returns a message element that reflects the current resource properties. As Figure 9 implies, the header includes a reference property to clearly identify the GAP instance.

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:gap="http://risc.uni_linz.ac.at/services/gap"
  xmlns:gap-pr=
 "http://risc.uni-linz.ac.at/services/gap/gap-properties.xsd">

  <Header xmlns:wsa="...">
    ...
    <wsa:To mustUnderstand="1">
        http://localhost:8080/wsrf/services/GapPort
    </wsa:To>
    <gap:ResourceIdentifier>1</gap:ResourceIdentifier>
  </Header>
  <Body>
    <wsrf-rp:GetResourcePropertyDocument/>
  </Body>
</Envelope>
```

Figure 9: Request resource properties

The returned message element is of type "GetResourcePropertyDocumentResponse" (see Figure 10) and contains all resource property elements according to the schema definition. The WSRF specification not only provides message elements and operations for retrieving property elements. The following list gives a short and not comprehensive overview of possible operations for manipulating properties and its subordinate value elements.

GetResourceProperty Retrieves one ore more value elements for the specified resource property element.

UpdateResourceProperties Changes the value of the specified resource property element.

InsertResourceProperties Adds a new value element of the specified resource property element.

DeleteResourceProperties Deletes a value element of the specified resource property element.

PutResourcePropertyDocument Sends a new version of a resource properties document to the WSRF web service.

The WSRF-specific message elements and its related operations are extensively described in the WSRF-RP specification document [10].

```
<Envelope>
  ...
  <Body>
   <wsrf:GetResourcePropertyDocumentResponse
       xmlns:wsrf="...">
     <gap:GapResourceProperties xmlns:gap="...">
       <wsrf:CurrentTime>
          2005-12-09T11:12:51.056+01:00
       </wsrf:CurrentTime>
       <wsrf:TerminationTime/>
       <gap:State>ready</gap:State>
       <gap:LastAction>no action</gap:LastAction>
       <gap:Reference>
          ...
       </gap:Reference>
     </gap:GapResourceProperties>
   </wsrf:GetResourcePropertyDocumentResponse>
  </Body>
</Envelope>
```

Figure 10: Response resource properties

Other than the mentioned operations for resource property access and manipulation the *GAPService* provides additional service-specific operations to deal with more GAP-related interactions. One simple method allows to send a plain GAP command to a created instance. In Figure 12 the service-specific message element "ExecuteActionRequest" contains such a command, that is brokered to the matching method and respectively to the GAP instance. In Appendix 8.1 the schema definition for this custom message element is given.

```
  ...
  <Body>
    <gap-pr:ExecuteActionRequest>
```

```
    <gap-pr:Action>PrevPrimeInt(911^11)</gap-pr:Action>
  </gap-pr:ExecuteActionRequest>
</Body>
```

Figure 11: Execute a GAP command

Not only custom schema definitions can be utilized for the use within WSRF web services. External schema definitions fit into the resource properties document as well. This strong feature virtually allows to include any information model that is expressible in an XML schema. By definition the *GAPService* is capable of processing OpenMath objects. OpenMath [20] is an XML-based notation for representing the semantics of mathematical objects. In Figure the term `(45 + (4 * 32)) - 98` is encoded as OpenMath object and is packed into the message element "ExecuteOMDocumentRequest". The underlying method uses so called phrasebooks to transform OpenMath objects to software-specific commands.

```
...
<Body>
  <gap-pr:ExecuteOMDocumentRequest>
    <gap-pr:OMDocument>
      <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
        <om:OMA>
          <om:OMS cd="arith1" name="minus"/>
          <om:OMA>
            <om:OMS cd="arith1" name="plus"/>
            <om:OMI>45</om:OMI>
            <om:OMA>
              <om:OMS cd="arith1" name="times"/>
              <om:OMI>4</om:OMI>
              <om:OMI>32</om:OMI>
            </om:OMA>
          </om:OMA>
          <om:OMI>98</om:OMI>
        </om:OMA>
      </om:OMOBJ>
    </gap-pr:OMDocument>
  </gap-pr:ExecuteOMDocumentRequest>
</Body>
```

Figure 12: Execute an OpenMath document

4.3.3 The Termination of the Instance

The previous sections dealt with the definition of resource property elements, their relation to service-specific message elements, and the general and special issues about creation and interaction with GAP instances. In this section the termination of resources is discussed which bases on the same fundamental techniques. Namely WSRF-specific message elements, potentially enhanced with additional parameters, and EPR's reference properties are encoded in a

SOAP message that is brokered to the matching method. Figure 13 shows the message that is sent to the *GAPService* to terminate an instance.

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:gap="..."
   xmlns:gap-pr="...">

 <Header xmlns:wsa=
      "http://schemas.xmlsoap.org/ws/2004/03/addressing">
   ...
  <gap:ResourceIdentifier>1</gap:ResourceIdentifier>
 </Header>

 <Body>
 <wsrf-rl:Destroy/>
 </Body>
</Envelope>
```

Figure 13: Terminate a resource

5 Mathematical Service Interaction with WS-BPEL

Section 4 gave a comprehensive view of WSRF and its adequacy as base for mathematical web services. As previously discussed, mathematical Web services in general require multiple interactions during initialization, execution, and termination. To automate this basic interaction pattern a more process-oriented approach is desirable. With focus on web services the WS-BPEL [14] specification suites well to describe processes that involve web service invocations.

Section 5.1 introduces the main concepts and features around WS-BPEL by means of an example process definition. Section 5.2 gives an overview of the Twister engine and its adaption to the Apache WSRF [9] framework.

5.1 WS-BPEL

WS-BPEL is a notation for specifying business processes based that model the interaction between services based on WSDL. The next sections presents the main concepts of WS-BPEL using the process described in Appendix 8.3.

5.1.1 Partner link types and partner links

Although a WS-BPEL process can be used for defining any process-oriented scenario, the specification mainly evolved out of the need for "cross-enterprise business interactions" [15]. The relationship between a process and an involved partner is usually a two-way dependency. The partner is not only the consumer of a service provided by a process, but also often offers a service to the process, especially when an asynchronous communication exists. To model such two-way relationships the concept of *partner link types* and *partner links* has been introduced.

A partner link type characterizes the relationship between two services by defining the role each plays during interaction within a process execution. It has to be defined in the corresponding WSDL document. Every partner link type can have one or two roles and for each role a port type is specified.

```
<wsdl:definitions>
  ...
  <plnk:partnerLinkType name="gapServiceLT">
    <plnk:role name="service">
      <plnk:portType name="gap:GapPort"/>
    </plnk:role>
  </plnk:partnerLinkType>
  ...
</wsdl:defintions>
```

Figure 14: Defintion of partner link types

Figure 14 shows a definition of a partner link type in GapService.wsdl (see Appendix 8.2 for full documentation). In this WSDL document only one role is defined, because port type *GapPort* and the corresponding *GAPService* do not take part in a two-way relation with any other service. *GAPService* only offers functionality to the process and does not play any other role.

The services with which a process interact are modeled as partner links. Each partner link has a unique name, a partner link type and a role. The role of the process is specified by the attribute `myRole` and the role of the partner is specified by the attribute `partnerRole`. When the partner link type defined in the WSDL document has only one role, one of these attributes can be skipped. Figure 15 presents the partner link definitions used in the example process.

```
<partnerLinks>
  <partnerLink name="gapInitiator"
    partnerLinkType="gap:initiatorLT" myRole="initiator"/>
  <partnerLink name="gapService"
    partnerLinkType="gap:gapServiceLT"
    partnerRole="service"/>
  <partnerLink name="gapFinisher"
    partnerLinkType="gap:finisherLT" myRole="finisher"/>
</partnerLinks>
```

Figure 15: Defintion of partner links

5.1.2 Correlation Sets

During the life cycle of a process instance it is typically the case that the instance participates in stateful conversations with involved partners. Any incoming message has to be delivered to the matching process instance. In WS-BPEL no dedicated technique for determine the appropriate instance exists, but instead an instance can be identified by defined data fields within received message elements. These sets of data fields that identify an process instance are called correlation sets and allow to determine to which conversation a message belongs.

Each correlation set has a name associated with it and is composed of properties. A property is a named and typed element which is defined in a WSDL document. A property alias is a mapping from a property to data field which is extracted from a WSDL message by applying a message-specific XPath [17] expression.

```
<bpel:property name="ResourceIdentifier" type="xsd:string"/>
<bpel:propertyAlias propertyName="ResourceIdentifier"
    messageType="gap:CreateInstanceResponse"
    part="gap-pr:CreateInstanceResponse"
    query="/ReferenceProperties/ResourceIdentifier"/>
```

The code fragment above shows the definition of a property and a property alias. In a process definition property aliases are used to define correlation sets as shown in the next figure. This correlation set can be referenced in a basic

activity and can be used by the process engine to identify the current process or any other stateful resource.

```
<correlationSets>
 <correlationSet name="resId" properties="ResourceIdentifier"/>
</correlationSets>
```

The property alias and the correlation set defined above are used by the Twister engine to extract the resource identifier (for more information see Section 5.2).

5.1.3 Variables and Assignments

Processes model the stateful interactions between the involved partners. Both, messages as defined by the web services' WSDL documents as well as intermediate data reflect the state of a process.

The concept of variables allows a process to hold such messages and to cope with state related information that is not intended for sending to partners. The type of a variable can either be a WSDL message or an arbitrary XML structure defined by a schema.

```
<variables>
 <variable name="action"
           messageType="gap:ExecuteActionRequest"/>
 <variable name="create"
           messageType="gap:CreateInstanceRequest"/>
 ...
</variables>
...
<assign>
 <copy>
  <from variable="input" part="msg"/>
  <to variable="action"
      part="gap-pr:ExecuteActionRequest"
      query="/gap-pr:Action"/>
 </copy>
</assign>
<assign>
 <copy>
  <from></from>
  <to variable="create"
      query="/gap-pr:CreateInstanceRequest"/>
 </copy>
</assign>
```

Figure 16: Definition and assignment of process variables

Figure 16 presents several variable definitions. A variable has a unique name and a message type, which is either defined in a WSDL document or in a XML schema definition. Passing data between partners, namely copying it from one to another variable, is a common task in a process. The *assign* activity enables

the manipulation of variables and the composition of new ones using expressions. In Figure 16 two assignments show this feature. The first assignment takes the *input* variable and copies the value from the *msg* element to the *action* variable. For example using the GAP command `PrevPrimeInt(911 Î1)` as *msg* value the *action* variable would have the following structure:

```
<gap-pr:ExecuteActionRequest>
  <gap-pr:Action>PrevPrimeInt(911^11)</gap-pr:Action>
<gap-pr:ExecuteActionRequest>
```

Refering to Figure 12, this newly constructed XML fragment is usable as message. The second assignment creates an empty element as specified in the *query* attribute. The *part* and *query* attributes can be used for composing arbitrary XML-based messages that are used for invoking web services or further processing.

5.1.4 Basic Activities

Basic activities handle the message flow between the involved partners. They allow to start process execution, to invoke web services and to terminate a process. Three types of basic activities exist: receive, invoke and reply.

A *receive* activity is the first basic activity in a process. It instantiates a process and accepts the first sent message for further operations. The *invoke* activity allows synchronous and asynchronous web service invocations. This type of activity specifies the port type and the related operation to be invoked. In case of a synchronous call the attribute `inputVariable` holds the message to be sent, the attribute `outputVariable` holds the received message after successful invocation. During an asynchronous call the `outputVariable` do not have to be defined. The following code sample is taken from Appendix 8.3 and shows an *invoke* operation.

```
<invoke partnerLink="gapService" portType="gap:GapPort"
   operation="executeAction" inputVariable="action"
   outputVariable="actionResponse">
</invoke>
```

A *reply* activity is the last basic activity during a process execution and returns the resulting message to the calling client. By definition of a WS-BPEL process it has to expose its functionality as a web service. The process provides its functionality to the client through *receive* and corresponding *reply* activities, that internally mark the initialization and the termination of a process.

5.1.5 Structured Activities

Structured activities bundle basic activities to express control patterns, the message flow and the coordination of message exchange. Several types of structured activities allow to model the process behavior.

A *sequence* prescribe the order in which activities take place. A sequence completes when the last activity has been performed. The process described in Appendix 8.3 applies a simple but fundamental usage pattern for a sequence. A receive activity is waiting for a client to invoke the process by sending a message. The received message can be manipulated before using it in additional web service invocations. After successful execution a reply activity sends the resulting message back to the client.

Additional structured activities, as known from higher programming languages, are *while* and *switch*. The *while* statement models a loop and therefore supports the repeated performance of specific activities as long as the given loop condition holds. Depending on a condition a *switch* activity offers different execution paths. In addition to the previously described constructs for specifying sequential execution, the *flow* activity creates a set of concurrent activities. The synchronization dependencies between the nested activities can be modeled by additional constructs. A more comprehensive view about structured activities is given in the specification [15].

5.2 The Twister Engine

The Twister engine [18] is an open source implementation of the WS-BPEL specification written in Java. It runs in a servlet container and bases on several other open source projects. The engine allows to deploy and execute processes, exposes the basic engine functionality as web services and offers a web-based user interface for administration purpose. As described in the architecture guide of [18] the Twister engine consists of 6 modules:

- The User Management module is used by Twister to manage users, groups and roles.
- The Worklist Manager module is used for registering and managing tasks.
- The Process Engine executes processes according to received messages.
- The Process Deployer parses and validates WSDL and process definitions and deploys this documents in the engine.
- The Client API encapsulates Twister's components providing an API for a Java client application.
- The Web Service Adapter exposes Twister components as web services.
- The Web module provides a user interfaces for administrating Twister's components.

The two last modules are directly deployed in the servlet container to encapsulate the rest of the provided functionality. The architecture is highly configurable and even allows to plugin in new activities, that are originally not defined by the WS-BPEL specification. The file *twister-configuration.xml* contains the main properties for configurating the Twister engine and its components.

All invocations made by the process engine are delegated to the MessageBroker as defined in *twister-configuration.xml* MessageBroker is an abstract class that receives all messages from the engine and redirect them to the methods of the subclass. To write a custom message broker the two following methods for synchronous and asynchronous operations have to be implemented:

```
protected void asyncSend(String partner, String portType,
    String operation, Document message);
protected Document syncSend(String partner, String portType,
    String operation, Document message);
```

The Twister engine is equipped with a default message broker for invoking classic web services based on SOAP and WSDL. This default message broker is not capable of dealing with WSRF-conform web services. Mainly the lack of support for the WS-Addressing specification forces to implement a new message broker.

To clearly identify a certain GAP instance when communicating with the *GAPService*, the SOAP header has to include a reference property that holds the resource identifier. The default message broker only invokes the defined web service but is not able to add additional information to the header. The new implemented message broker applies the extended WSRF approach. It extracts the value of the resource identifier from a defined correlation set (see Section 5.1.2) and adds it as reference property to the SOAP header. The two listings in Section 5.1.2 show how to define a clear relation between the received message after GAP instance creation (as described in Section 4.3.1) and the instance's resource identifier. Due to the definition of this property alias and its corresponding correlation set the Twister engine is capable to directly access the value of the resource identifier.

6 Processing MSDL Descriptions

This section presents the underlying information model of MSDL and the process of transforming MSDL descriptions to WS-BPEL documents.

6.1 The Information Model of MSDL

That mathematical services can be discovered by clients, they need to advertise their capabilities in a machine-understandable way. MSDL[23, 24] was developed in the frame of the "MathBroker" project [19] with influences from the MONET project [21]. Figure 17 shows a sample MSDL description for the *GAPService*. This sample description relies on the extended MSDL version [25] as proposed by the MONET project in [28]. We have chosen this version because of the possibility to formulate a basic execution behaviour. The information model, which is conceptually similar to the original "MathBroker" model, consists of the following parts:

Classification The classification specifies the service in terms of the problem to be solved, referenced taxanomies, semantic descriptions, and supported directives (e.g. find, prove, lookup, ...).

Implementation The implementation provides details about the soft- and hardware utilized by the service. Additionally the needed actions for solving the problem can be definied.

Service Interface Description This entity specifies the static interface description, which is typically a WSDL document.

Service Binding The service binding contains the mapping from problem components and actions to elements in the referenced WSDL document.

Service Broker Interface This is the interface exposed to a possible broker. It typically consists of a service URI and a service description. The usage of a broker is not prescribed in the MONET project and is left open to extened the architecture.

```
<definitions>

<service name="GapPort">
 <classification>
  <problem href="http://monet.nag.co.uk/problems/Problem"/>
  <directive-type href="http://monet.nag.co.uk/owl#lookup"/>
 </classification>
 <implementation>
  <software href ="http://monet.nag.co.uk/owl#GAP"/>
  <hardware href ="http://monet.nag.co.uk/owl#PentiumSystem"/>
  <action role="Initialize" name="create"/>
  <action role="Execute" name="compute"/>
  <action role="Terminate" name="destroy"/>
 </implementation>
 <service-interface-description
      href="http://localhost:8080/wsrf/services/GapPort?wsdl"/>

 <service-binding>
  <map action="create" operation="gap:createInstance"/>
  <message-construction
      io-ref="gap-pr:CreateInstanceRequest"
      message-name="gap:CreateInstanceRequest"
      message-part="CreateInstanceRequest"/>
  <message-construction
      io-ref="gap-pr:CreateInstanceResponse"
      message-name="gap:CreateInstanceResponse"
      message-part="CreateInstanceResponse"/>
      ...
 </service-binding>
```

```
  <broker-interface>
    ...
  </broker-interface>

</service>

</definitions>
```

Figure 17: An example MSDL document

Figure 17 shows a fragment of a MSDL document that specifies the *GAPService*. In Appendix 8.4 the complete MSDL document is described.

6.2 Transforming MSDL to WS-BPEL

To potentially include existing MSDL documents in our future work, we have implemented a software that transforms MSDL documents to WS-BPEL documents. The transformation process mainly relies on the basic execution behavior described in the *implementation* and *service-binding* elements of MSDL.

Sometimes a service requires several steps for solving a mathematical problem (see the introductory example in Section 3) and this requirement should be made explicit. In MSDL this is handled in two stages:

- First a sequence of actions is defined within the *implementation* element. Every action has an assigned role (namely "Initialize", "Execute" or "Terminate") to provide semantic information about the basic execution behavior.
- Second these abstract actions are mapped to concrete service operations and messages as defined in the *service-binding* element. The *map* element maps abstract actions to service operations that are defined in a WSDL document. The *message-construction* element describes how to construct and deconstruct messages understood by the service in terms of problem descriptions.

The presented method for describing the basic execution behavior in MSDL notation has several weaknesses. First, in relation to WS-BPEL only an execution sequence of predefined actions can be formulated. Second, the role names that are assigned to the abstract actions are plain text strings. An URI reference to an unambiguous concept (that is e.g. defined in an OWL ontology) would be a more adequate way to describe the different types of abstract actions. The last and most apparent deficiencies deal with message construction. As web service technologies evolved in the past years, the use of a document-oriented binding style for SOAP messages has been widely accepted (in contrary to the "classic" RPC-oriented binding style). Unfortunately, the MSDL specification seems to mainly support RPC-oriented web services. So it would be feasible to explicitly define the XML schema type of a message in the *message-construct* element to support document-oriented web services as well. Additionally, there does not exist a well-defined relationship between *map* elements and *message-construct*

elements nor does these elements define the general type of exchanged messages (e.g. input or output messages).

Due to mentioned weaknesses we had to make several assumptions about MSDL elements and its structure, especially about message construction, to generate a proper WS-BPEL document.

- Every abstract action defined in the *implementation* element has a corresponding *invoke* activity in the generated WS-BPEL document.
- Every *map* element is followed by exactly two *message-construct* elements that belong to this mapped service operation. The first *message-construct* reflects the input message, the second *message-construct* reflects the output message of the mapped service operation.
- Originally the *io-ref* attribute in the *message-construct* element is used for referencing a problem description. We used it for specifying the XML schema type of a message. This additional information is needed for generating the *variable* declarations and the *assign* activities in the WS-BPEL document.

The software that handles the MSDL transformation is based on XMLBeans [32] and the Velocity Engine [33]. XMLBeans is a framework that allows to bind XML elements to Java objects. The Velocity Engine is a Java-based template engine that provides a simple but powerful template language which allows to reference Java objects. XMLBeans internally represent the MSDL document as set of objects that are used by the Velocity Template Engine to generate a WS-BPEL document. As the execution environment described above, the transforming software can be evaluated at our demonstration site.

7 Conclusion

In this paper we have presented an execution environment for mathematical services based on WSRF and WS-BPEL. We have combined these current web service standards to introduce a dynamic approach for executing mathematical web services. Additionally we have presented an approach to include existing MSDL document in our current research. Several additional aspects such as finding and querying matching mathematical services that can solve a mathematical problem are not part of this work, but are covered elsewhere [29].

The presented execution environment allows to execute manually created process descriptions. Future work will concentrate on the automated derivation of interaction protocols from semantic service descriptions. The Semantic Web community provides several technologies and standards such as OWL-S [30] or WSMO [31], but further investigation is needed to find an appropriate candidate technology for our purpose.

8 Appendix

8.1 gap-properties.xsd

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:gap-pr="http://risc.uni-linz.ac.at/services/gap/gap-properties.xsd"
 targetNamespace="http://risc.uni-linz.ac.at/services/gap/gap-properties.xsd"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:wsrf-rl="http://docs.oasis-open.org/wsrf/2004/11/
   wsrf-WS-ResourceLifetime-1.2-draft-04.xsd"
 xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/2004/11/
   wsrf-WS-BaseFaults-1.2-draft-03.xsd"
 xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/2004/11/
   wsrf-WS-ResourceProperties-1.2-draft-05.xsd"
 xmlns:om="http://www.openmath.org/OpenMath"
 attributeFormDefault="unqualified" elementFormDefault="qualified">

 <xsd:import namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  schemaLocation="../spec/wsa/WS-Addressing-2004_08_10.xsd"/>
 <xsd:import namespace="http://docs.oasis-open.org/wsrf/2004/11/
   wsrf-WS-ResourceLifetime-1.2-draft-04.xsd"
  schemaLocation="../spec/wsrf/WS-ResourceLifetime-1_2-Draft_04.xsd"/>
 <xsd:import
  namespace="http://docs.oasis-open.org/wsrf/2004/11/
   wsrf-WS-BaseFaults-1.2-draft-03.xsd"
  schemaLocation="../spec/wsrf/WS-BaseFaults-1_2-Draft_03.xsd"/>
 <xsd:import
  namespace="http://docs.oasis-open.org/wsrf/2004/11/
   wsrf-WS-ResourceProperties-1.2-draft-05.xsd"
  schemaLocation="../spec/wsrf/WS-ResourceProperties-1_2-Draft_05.xsd"/>
 <xsd:import namespace="http://www.openmath.org/OpenMath"
  schemaLocation="../wsdl/openmath2.xsd"/>

 <xsd:element name="GapResourceProperties">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="gap-pr:Reference"/>
    <xsd:element ref="gap-pr:State"/>
    <xsd:element ref="gap-pr:LastAction"/>
    <xsd:element ref="wsrf-rl:CurrentTime"/>
    <xsd:element ref="wsrf-rl:TerminationTime"/>
    <xsd:element ref="wsrf-rp:QueryExpressionDialect" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="Reference" type="wsa:EndpointReferenceType"/>
 <xsd:element name="State" type="gap-pr:StateType"/>
  <xsd:simpleType name="StateType">
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="ready"/>
    <xsd:enumeration value="processing"/>
    <xsd:enumeration value="aborted"/>
    <xsd:enumeration value="completed"/>
   </xsd:restriction>
  </xsd:simpleType>
 <xsd:element name="LastAction" type="xsd:string"/>

 <!-- Custom operation messages -->
 <xsd:element name="CreateInstanceRequest"/>
 <xsd:element name="CreateInstanceResponse">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="gap-pr:Reference"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
```

```
<xsd:element name="ExecuteActionRequest">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="gap-pr:Action"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name="ExecuteActionResponse">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="gap-pr:Result"/>
   <xsd:element ref="gap-pr:Reference"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Action" type="xsd:string"/>
<xsd:element name="Result" type="xsd:string"/>

<xsd:element name="ExecuteOMDocumentRequest">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="gap-pr:OMDocument"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name="ExecuteOMDocumentResponse">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="gap-pr:OMDocument"/>
   <xsd:element ref="gap-pr:Reference"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element name="OMDocument">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:any maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>
       </xsd:element>

<!-- Custom errors -->
<xsd:element name="GapFault">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="wsrf-bf:BaseFaultType"/>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>

<xsd:element name="PhrasebookFault">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="wsrf-bf:BaseFaultType"/>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
<xsd:element name="GeneralClientFault">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="wsrf-bf:BaseFaultType"/>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>

</xsd:schema>
```

8.2 GapService.wsdl

```
<?xml version="1.0"?>
<wsdl:definitions name="GapResourceDefinition"
 targetNamespace="http://risc.uni-linz.ac.at/services/gap"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/2004/11/
␣␣wsrf-WS-ResourceProperties-1.2-draft-05.xsd"
 xmlns:wsrf-rpw="http://docs.oasis-open.org/wsrf/2004/11/
␣␣wsrf-WS-ResourceProperties-1.2-draft-05.wsdl"
 xmlns:wsrf-rlw="http://docs.oasis-open.org/wsrf/2004/11/
␣␣wsrf-WS-ResourceLifetime-1.2-draft-04.wsdl"
 xmlns:gap="http://risc.uni-linz.ac.at/services/gap"
 xmlns:gap-pr="http://risc.uni-linz.ac.at/services/gap/gap-properties.xsd"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

 <wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/11/
␣␣␣wsrf-WS-ResourceProperties-1.2-draft-05.wsdl"
   location="../spec/wsrf/WS-ResourceProperties-1_2-Draft_05.wsdl"/>
 <wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/11/
␣␣␣wsrf-WS-ResourceLifetime-1.2-draft-04.wsdl"
   location="../spec/wsrf/WS-ResourceLifetime-1_2-Draft_04.wsdl"/>

 <wsdl:types>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xsd:import namespace="http://risc.uni-linz.ac.at/services/gap/gap-properties.xsd"
      schemaLocation="../wsdl/gap-properties.xsd"/>
  </xsd:schema>
 </wsdl:types>

 <!-- ================ Message Definitions for Custom Operations ================ -->
 <wsdl:message name="CreateInstanceRequest">
  <wsdl:part name="CreateInstanceRequest" element="gap-pr:CreateInstanceRequest"/>
 </wsdl:message>
 <wsdl:message name="CreateInstanceResponse">
  <wsdl:part name="CreateInstanceResponse" element="gap-pr:CreateInstanceResponse"/>
 </wsdl:message>

 <wsdl:message name="ExecuteActionRequest">
  <wsdl:part name="ExecuteActionRequest" element="gap-pr:ExecuteActionRequest"/>
 </wsdl:message>
 <wsdl:message name="ExecuteActionResponse">
  <wsdl:part name="ExecuteActionResponse" element="gap-pr:ExecuteActionResponse"/>
 </wsdl:message>

 <wsdl:message name="ExecuteOMDocumentRequest">
  <wsdl:part name="ExecuteOMDocumentRequest" element="gap-pr:ExecuteOMDocumentRequest"/>
 </wsdl:message>
 <wsdl:message name="ExecuteOMDocumentResponse">
  <wsdl:part name="ExecuteOMDocumentResponse" element="gap-pr:ExecuteOMDocumentResponse"/>
 </wsdl:message>

 <message name="GapFault">
  <wsdl:part name="GapFault" element="gap-pr:GapFault"/>
 </message>
 <message name="PhrasebookFault">
  <wsdl:part name="PhrasebookFault" element="gap-pr:PhrasebookFault"/>
 </message>
 <message name="GeneralClientFault">
  <wsdl:part name="GeneralClientFault" element="gap-pr:GeneralClientFault"/>
 </message>

 <wsdl:portType name="GapPortType"
       wsrf-rp:ResourceProperties="gap-pr:GapResourceProperties">
  <wsdl:operation name="GetResourcePropertyDocument">
   <wsdl:input name="GetResourcePropertyDocumentRequest"
```

```
      message="wsrf-rpw:GetResourcePropertyDocumentRequest"/>
    <wsdl:output name="GetResourcePropertyDocumentResponse"
      message="wsrf-rpw:GetResourcePropertyDocumentResponse"/>
    <wsdl:fault name="ResourceUnknownFault"
      message="wsrf-rpw:ResourceUnknownFault"/>
  </wsdl:operation>
  <wsdl:operation name="GetResourceProperty">
    <wsdl:input name="GetResourcePropertyRequest"
      message="wsrf-rpw:GetResourcePropertyRequest"/>
    <wsdl:output name="GetResourcePropertyResponse"
      message="wsrf-rpw:GetResourcePropertyResponse"/>
    <wsdl:fault name="ResourceUnknownFault"
      message="wsrf-rpw:ResourceUnknownFault"/>
    <wsdl:fault name="InvalidResourcePropertyQNameFault"
      message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
  </wsdl:operation>
  <wsdl:operation name="GetMultipleResourceProperties">
    <wsdl:input name="GetMultipleResourcePropertiesRequest"
      message="wsrf-rpw:GetMultipleResourcePropertiesRequest"/>
    <wsdl:output name="GetMultipleResourcePropertiesResponse"
      message="wsrf-rpw:GetMultipleResourcePropertiesResponse"/>
    <wsdl:fault name="ResourceUnknownFault"
      message="wsrf-rpw:ResourceUnknownFault"/>
    <wsdl:fault name="InvalidResourcePropertyQNameFault"
      message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
  </wsdl:operation>
  <wsdl:operation name="SetResourceProperties">
    <wsdl:input name="SetResourcePropertiesRequest"
      message="wsrf-rpw:SetResourcePropertiesRequest"/>
    <wsdl:output name="SetResourcePropertiesResponse"
      message="wsrf-rpw:SetResourcePropertiesResponse"/>
    <wsdl:fault name="ResourceUnknownFault"
      message="wsrf-rpw:ResourceUnknownFault"/>
    <wsdl:fault name="InvalidSetResourcePropertiesRequestContentFault"
      message="wsrf-rpw:InvalidSetResourcePropertiesRequestContentFault"/>
    <wsdl:fault name="UnableToModifyResourcePropertyFault"
      message="wsrf-rpw:UnableToModifyResourcePropertyFault"/>
    <wsdl:fault name="InvalidResourcePropertyQNameFault"
      message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
    <wsdl:fault name="SetResourcePropertyRequestFailedFault"
      message="wsrf-rpw:SetResourcePropertyRequestFailedFault"/>
  </wsdl:operation>
  <wsdl:operation name="InsertResourceProperties">
    <wsdl:input name="InsertResourcePropertiesRequest"
      message="wsrf-rpw:InsertResourcePropertiesRequest"/>
    <wsdl:output name="InsertResourcePropertiesResponse"
      message="wsrf-rpw:InsertResourcePropertiesResponse"/>
    <wsdl:fault name="ResourceUnknownFault"
      message="wsrf-rpw:ResourceUnknownFault"/>
    <wsdl:fault name="InvalidInsertResourcePropertiesRequestContentFault"
      message="wsrf-rpw:InvalidInsertResourcePropertiesRequestContentFault"/>
    <wsdl:fault name="UnableToModifyResourcePropertyFault"
      message="wsrf-rpw:UnableToModifyResourcePropertyFault"/>
    <wsdl:fault name="InvalidResourcePropertyQNameFault"
      message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
    <wsdl:fault name="InsertResourcePropertyRequestFailedFault"
      message="wsrf-rpw:InsertResourcePropertyRequestFailedFault"/>
  </wsdl:operation>
  <wsdl:operation name="UpdateResourceProperties">
    <wsdl:input name="UpdateResourcePropertiesRequest"
      message="wsrf-rpw:UpdateResourcePropertiesRequest"/>
    <wsdl:output name="UpdateResourcePropertiesResponse"
      message="wsrf-rpw:UpdateResourcePropertiesResponse"/>
    <wsdl:fault name="ResourceUnknownFault"
      message="wsrf-rpw:ResourceUnknownFault"/>
    <wsdl:fault name="InvalidUpdateResourcePropertiesRequestContentFault"
      message="wsrf-rpw:InvalidUpdateResourcePropertiesRequestContentFault"/>
    <wsdl:fault name="UnableToModifyResourcePropertyFault"
```

```
    message="wsrf-rpw:UnableToModifyResourcePropertyFault"/>
  <wsdl:fault name="InvalidResourcePropertyQNameFault"
    message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
  <wsdl:fault name="UpdateResourcePropertyRequestFailedFault"
    message="wsrf-rpw:UpdateResourcePropertyRequestFailedFault"/>
 </wsdl:operation>
 <wsdl:operation name="DeleteResourceProperties">
  <wsdl:input name="DeleteResourcePropertiesRequest"
    message="wsrf-rpw:DeleteResourcePropertiesRequest"/>
  <wsdl:output name="DeleteResourcePropertiesResponse"
    message="wsrf-rpw:DeleteResourcePropertiesResponse"/>
  <wsdl:fault name="ResourceUnknownFault"
    message="wsrf-rpw:ResourceUnknownFault"/>
  <wsdl:fault name="UnableToModifyResourcePropertyFault"
    message="wsrf-rpw:UnableToModifyResourcePropertyFault"/>
  <wsdl:fault name="InvalidResourcePropertyQNameFault"
    message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
  <wsdl:fault name="DeleteResourcePropertyRequestFailedFault"
    message="wsrf-rpw:DeleteResourcePropertyRequestFailedFault"/>
 </wsdl:operation>
 <wsdl:operation name="QueryResourceProperties">
  <wsdl:input name="QueryResourcePropertiesRequest"
    message="wsrf-rpw:QueryResourcePropertiesRequest"/>
  <wsdl:output name="QueryResourcePropertiesResponse"
    message="wsrf-rpw:QueryResourcePropertiesResponse"/>
  <wsdl:fault name="ResourceUnknownFault"
    message="wsrf-rpw:ResourceUnknownFault"/>
  <wsdl:fault name="InvalidResourcePropertyQNameFault"
    message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
  <wsdl:fault name="UnknownQueryExpressionDialectFault"
    message="wsrf-rpw:UnknownQueryExpressionDialectFault"/>
  <wsdl:fault name="InvalidQueryExpressionFault"
    message="wsrf-rpw:InvalidQueryExpressionFault"/>
  <wsdl:fault name="QueryEvaluationErrorFault"
    message="wsrf-rpw:QueryEvaluationErrorFault"/>
 </wsdl:operation>
 <wsdl:operation name="Destroy">
  <wsdl:input name="DestroyRequest"
    message="wsrf-rlw:DestroyRequest"/>
  <wsdl:output name="DestroyResponse"
    message="wsrf-rlw:DestroyResponse"/>
  <wsdl:fault name="ResourceNotDestroyedFault"
    message="wsrf-rlw:ResourceNotDestroyedFault"/>
  <wsdl:fault name="ResourceUnknownFault"
    message="wsrf-rlw:ResourceUnknownFault"/>
 </wsdl:operation>
 <wsdl:operation name="SetTerminationTime">
  <wsdl:input name="SetTerminationTimeRequest"
    message="wsrf-rlw:SetTerminationTimeRequest"/>
  <wsdl:output name="SetTerminationTimeResponse"
    message="wsrf-rlw:SetTerminationTimeResponse"/>
  <wsdl:fault name="UnableToSetTerminationTimeFault"
    message="wsrf-rlw:UnableToSetTerminationTimeFault"/>
  <wsdl:fault name="ResourceUnknownFault"
    message="wsrf-rlw:ResourceUnknownFault"/>
  <wsdl:fault name="TerminationTimeChangeRejectedFault"
    message="wsrf-rlw:TerminationTimeChangeRejectedFault"/>
 </wsdl:operation>

 <!-- custom operations -->
 <wsdl:operation name="createInstance">
  <wsdl:input  name="CreateInstanceRequest"
    message="gap:CreateInstanceRequest"/>
  <wsdl:output name="CreateInstanceResponse"
    message="gap:CreateInstanceResponse"/>
  <wsdl:fault name="GeneralClientFault"
    message="gap:GeneralClientFault"/>
 </wsdl:operation>
```

```
  <wsdl:operation name="executeAction">
   <wsdl:input  name="ExecuteActionRequest"
     message="gap:ExecuteActionRequest"/>
   <wsdl:output name="ExecuteActionResponse"
     message="gap:ExecuteActionResponse"/>
   <wsdl:fault name="GapFault"
     message="gap:GapFault"/>
   <wsdl:fault name="GeneralClientFault"
     message="gap:GeneralClientFault"/>
  </wsdl:operation>
  <wsdl:operation name="executeOMDocument">
   <wsdl:input  name="ExecuteActionRequest"
     message="gap:ExecuteOMDocumentRequest"/>
   <wsdl:output name="ExecuteActionResponse"
     message="gap:ExecuteOMDocumentResponse"/>
   <wsdl:fault name="PhrasebookFault"
     message="gap:PhrasebookFault"/>
   <wsdl:fault name="GeneralClientFault"
     message="gap:GeneralClientFault"/>
  </wsdl:operation>
 </wsdl:portType>

 <binding name="GapSoapHttpBinding" type="gap:GapPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetResourcePropertyDocument">
   <soap:operation
     soapAction="http://risc.uni-linz.ac.at/services/gap/GetResourcePropertyDocument"/>
   <input name="GetResourcePropertyDocumentRequest">
    <soap:body use="literal"/>
   </input>
   <output name="GetResourcePropertyDocumentResponse">
    <soap:body use="literal"/>
   </output>
   <fault name="ResourceUnknownFault">
    <soap:fault name="ResourceUnknownFault" use="literal"/>
   </fault>
  </operation>
  <operation name="GetResourceProperty">
   <soap:operation
     soapAction="http://risc.uni-linz.ac.at/services/gap/GetResourceProperty"/>
   <input name="GetResourcePropertyRequest">
    <soap:body use="literal"/>
   </input>
   <output name="GetResourcePropertyResponse">
    <soap:body use="literal"/>
   </output>
   <fault name="ResourceUnknownFault">
    <soap:fault name="ResourceUnknownFault" use="literal"/>
   </fault>
   <fault name="InvalidResourcePropertyQNameFault">
    <soap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
   </fault>
  </operation>
  <operation name="GetMultipleResourceProperties">
   <soap:operation
     soapAction="http://risc.uni-linz.ac.at/services/gap/GetMultipleResourceProperties"/>
   <input name="GetMultipleResourcePropertiesRequest">
    <soap:body use="literal"/>
   </input>
   <output name="GetMultipleResourcePropertiesResponse">
    <soap:body use="literal"/>
   </output>
   <fault name="ResourceUnknownFault">
    <soap:fault name="ResourceUnknownFault" use="literal"/>
   </fault>
   <fault name="InvalidResourcePropertyQNameFault">
    <soap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
   </fault>
```

```
</operation>
<operation name="SetResourceProperties">
 <soap:operation
   soapAction="http://risc.uni-linz.ac.at/services/gap/SetResourceProperties"/>
 <input name="SetResourcePropertiesRequest">
  <soap:body use="literal"/>
 </input>
 <output name="SetResourcePropertiesResponse">
  <soap:body use="literal"/>
 </output>
 <fault name="ResourceUnknownFault">
  <soap:fault name="ResourceUnknownFault" use="literal"/>
 </fault>
 <fault name="InvalidSetResourcePropertiesRequestContentFault">
  <soap:fault name="InvalidSetResourcePropertiesRequestContentFault" use="literal"/>
 </fault>
 <fault name="UnableToModifyResourcePropertyFault">
  <soap:fault name="UnableToModifyResourcePropertyFault" use="literal"/>
 </fault>
 <fault name="InvalidResourcePropertyQNameFault">
  <soap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
 </fault>
 <fault name="SetResourcePropertyRequestFailedFault">
  <soap:fault name="SetResourcePropertyRequestFailedFault" use="literal"/>
 </fault>
</operation>
<operation name="QueryResourceProperties">
 <soap:operation
   soapAction="http://risc.uni-linz.ac.at/services/gap/QueryResourceProperties"/>
 <input name="QueryResourcePropertiesRequest">
  <soap:body use="literal"/>
 </input>
 <output name="QueryResourcePropertiesResponse">
  <soap:body use="literal"/>
 </output>
 <fault name="ResourceUnknownFault">
  <soap:fault name="ResourceUnknownFault" use="literal"/>
 </fault>
 <fault name="InvalidResourcePropertyQNameFault">
  <soap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
 </fault>
 <fault name="UnknownQueryExpressionDialectFault">
  <soap:fault name="UnknownQueryExpressionDialectFault" use="literal"/>
 </fault>
 <fault name="InvalidQueryExpressionFault">
  <soap:fault name="InvalidQueryExpressionFault" use="literal"/>
 </fault>
 <fault name="QueryEvaluationErrorFault">
  <soap:fault name="QueryEvaluationErrorFault" use="literal"/>
 </fault>
</operation>
<operation name="Destroy">
 <soap:operation
   soapAction="http://risc.uni-linz.ac.at/services/gap/Destroy"/>
 <input name="DestroyRequest">
  <soap:body use="literal"/>
 </input>
 <output name="DestroyResponse">
  <soap:body use="literal"/>
 </output>
 <fault name="ResourceNotDestroyedFault">
  <soap:fault name="ResourceNotDestroyedFault" use="literal"/>
 </fault>
 <fault name="ResourceUnknownFault">
  <soap:fault name="ResourceUnknownFault" use="literal"/>
 </fault>
</operation>
<operation name="SetTerminationTime">
```

```
  <soap:operation
    soapAction="http://risc.uni-linz.ac.at/services/gap/SetTerminationTime"/>
  <input name="SetTerminationTimeRequest">
   <soap:body use="literal"/>
  </input>
  <output name="SetTerminationTimeResponse">
   <soap:body use="literal"/>
  </output>
  <fault name="UnableToSetTerminationTimeFault">
   <soap:fault name="UnableToSetTerminationTimeFault" use="literal"/>
  </fault>
  <fault name="ResourceUnknownFault">
   <soap:fault name="ResourceUnknownFault" use="literal"/>
  </fault>
  <fault name="TerminationTimeChangeRejectedFault">
   <soap:fault name="TerminationTimeChangeRejectedFault" use="literal"/>
  </fault>
 </operation>

 <!-- custom operations -->
 <operation name="createInstance">
  <soap:operation
    soapAction="http://risc.uni-linz.ac.at/services/gap/createInstance"/>
  <input>
   <soap:body use="literal"/>
  </input>
  <output>
   <soap:body use="literal"/>
  </output>
  <fault name="GeneralClientFault">
   <soap:fault name="GeneralClientFault" use="literal"/>
  </fault>
 </operation>
 <operation name="executeAction">
  <soap:operation
    soapAction="http://risc.uni-linz.ac.at/services/gap/executeAction"/>
  <input>
   <soap:body use="literal"/>
  </input>
  <output>
   <soap:body use="literal"/>
  </output>
  <fault name="GapFault">
   <soap:fault name="GapFault" use="literal"/>
  </fault>
  <fault name="GeneralClientFault">
   <soap:fault name="GeneralClientFault" use="literal"/>
  </fault>
 </operation>
 <operation name="executeOMDocument">
  <soap:operation
    soapAction="http://risc.uni-linz.ac.at/services/gap/executeOMDocument"/>
  <input>
   <soap:body use="literal"/>
  </input>
  <output>
   <soap:body use="literal"/>
  </output>
  <fault name="PhrasebookFault">
   <soap:fault name="PhrasebookFault" use="literal"/>
  </fault>
  <fault name="GeneralClientFault">
   <soap:fault name="GeneralClientFault" use="literal"/>
  </fault>
 </operation>
</binding>

<service name="GapService">
```

```
  <port name="GapPort" binding="gap:GapSoapHttpBinding">
   <soap:address location="http://dragonfly.risc.uni-linz.ac.at:8080/wsrf/services/gapservice"/>
  </port>
 </service>

 <plnk:partnerLinkType name="initiatorLT">
  <plnk:role name="initiator">
   <plnk:portType name="initiatorPT"/>
  </plnk:role>
 </plnk:partnerLinkType>
 <plnk:partnerLinkType name="gapServiceLT">
  <plnk:role name="service">
   <plnk:portType name="gap:GapPort"/>
  </plnk:role>
 </plnk:partnerLinkType>
 <plnk:partnerLinkType name="finisherLT">
  <plnk:role name="finisher">
   <plnk:portType name="finisherPT"/>
  </plnk:role>
 </plnk:partnerLinkType>

 <bpel:property name="ResourceIdentifier" type="xsd:string"/>
 <bpel:propertyAlias propertyName="ResourceIdentifier"
    messageType="gap:CreateInstanceResponse"
    part="gap-pr:CreateInstanceResponse"
    query="/ReferenceProperties/ResourceIdentifier"/>

</wsdl:definitions>
```

8.3 create-gap-instance.bpel

```
<?xml version="1.0" encoding="UTF-8"?>

<process name="gapProcess"
 targetNamespace="http://risc.uni-linz.ac.at/processes/gap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
 xmlns:gap="http://risc.uni-linz.ac.at/services/gap"
 xmlns:gap-pr="http://risc.uni-linz.ac.at/services/gap/gap-properties.xsd"
 xmlns:tr="http://risc.uni-linz.ac.at/services/transform"
 xmlns:tr-pr="http://risc.uni-linz.ac.at/services/transform/transform-properties.xsd"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/2004/11/
   wsrf-WS-ResourceProperties-1.2-draft-05.xsd"
 xmlns:wsrf-rpw="http://docs.oasis-open.org/wsrf/2004/11/
   wsrf-WS-ResourceProperties-1.2-draft-05.wsdl"
 xmlns:wsrf-rlw="http://docs.oasis-open.org/wsrf/2004/11/
   wsrf-WS-ResourceLifetime-1.2-draft-04.wsdl">

 <partnerLinks>
  <partnerLink name="gapInitiator" partnerLinkType="gap:initiatorLT" myRole="initiator"/>
  <partnerLink name="gapService" partnerLinkType="gap:gapServiceLT" partnerRole="service"/>
  <partnerLink name="gapFinisher" partnerLinkType="gap:finisherLT" myRole="finisher"/>
 </partnerLinks>

 <variables>
  <variable name="resourceID" messageType="xsd:string"/>
  <variable name="input" messageType="xsd:string"/>
  <variable name="action" messageType="gap:ExecuteActionRequest"/>
  <variable name="actionResponse" messageType="gap:ExecuteActionResponse"/>
  <variable name="create" messageType="gap:CreateInstanceRequest"/>
  <variable name="createResponse" messageType="gap:CreateInstanceResponse"/>
  <variable name="destroy" messageType="wsrf-rlw:DestroyRequest"/>
  <variable name="destroyResponse" messageType="wsrf-rlw:DestroyResponse"/>
  <variable name="output" messageType="xsd:string"/>
 </variables>
```

```
 <correlationSets>
  <correlationSet name="resId" properties="ResourceIdentifier"/>
 </correlationSets>

 <sequence>
  <receive partnerLink="gapInitiator" portType="initiatorPT" operation="initiate"
    variable="input" createInstance="yes"/>

  <assign>
   <copy>
    <from variable="input" part="msg"/>
    <to variable="action" part="gap-pr:ExecuteActionRequest" query="/gap-pr:Action"/>
   </copy>
  </assign>
  <assign>
   <copy>
    <from></from>
    <to variable="create" query="/gap-pr:CreateInstanceRequest"/>
   </copy>
  </assign>
  <invoke partnerLink="gapService" portType="gap:GapPort" operation="createInstance"
    inputVariable="create" outputVariable="createResponse">
   <correlations>
    <correlation set="resId" pattern="in" initiate="yes"/>
   </correlations>
  </invoke>

  <assign>
   <copy>
    <from variable="createResponse" part="CreateInstanceResponse"
      query="/ReferenceProperties/ResourceIdentifier"></from>
    <to variable="resourceID" query="/gap:ResourceIdentfier"/>
   </copy>
  </assign>

  <invoke partnerLink="gapService" portType="gap:GapPort" operation="executeAction"
    inputVariable="action" outputVariable="actionResponse">
  </invoke>

  <assign>
   <copy>
    <from variable="createResponse" part="CreateInstanceResponse"
      query="/ReferenceProperties/ResourceIdentifier"></from>
    <to variable="resourceID" query="/gap:ResourceIdentifier"/>
   </copy>
  </assign>
  <assign>
   <copy>
    <from></from>
    <to variable="destroy" query="/wsrf-rl:Destroy"/>
   </copy>
  </assign>
  <invoke partnerLink="gapService" portType="gap:GapPort" operation="Destroy"
    inputVariable="destroy" outputVariable="destroyResponse">
  </invoke>

 </sequence>
</process>
```

8.4 GapService.msdl

```
<definitions targetNamespace="http://www.orcca.on.ca/MONET/samples/msdl"
 xmlns="http://monet.nag.co.uk/monet/ns"
 xmlns:wsrf-rl="http://docs.oasis-open.org/wsrf/2004/11/
   wsrf-WS-ResourceLifetime-1.2-draft-04.xsd"
 xmlns:wsrl-rlw="http://docs.oasis-open.org/wsrf/2004/11/
   wsrf-WS-ResourceLifetime-1.2-draft-04.wsdl"
```

```
 xmlns:gap="http://risc.uni-linz.ac.at/services/gap"
 xmlns:gap-pr="http://risc.uni-linz.ac.at/services/gap/gap-properties.xsd">

<service name="GapPort">
 <classification>
  <problem href="http://monet.nag.co.uk/problems/Problem"/>
  <directive-type href="http://monet.nag.co.uk/owl#lookup"/>
 </classification>
 <implementation>
  <software href ="http://monet.nag.co.uk/owl#GAP"/>
  <hardware href ="http://monet.nag.co.uk/owl#PentiumSystem"/>
  <action role="Initialize" name="create"/>
  <action role="Execute" name="lookup"/>
  <action role="Terminate" name="destroy"/>
 </implementation>
 <service-interface-description
    href="http://dragonfly.risc.uni-linz.ac.at:8080/wsrf/services/GapPort?wsdl"/>

 <service-binding>
  <map action="create" operation="gap:createInstance"/>
  <message-construction io-ref="gap-pr:CreateInstanceRequest"
       message-name="gap:CreateInstanceRequest" message-part="CreateInstanceRequest"/>
  <message-construction io-ref="gap-pr:CreateInstanceResponse"
       message-name="gap:CreateInstanceResponse" message-part="CreateInstanceResponse"/>

  <map action="lookup" operation="gap:executeAction"/>
  <message-construction io-ref="gap-pr:ExecuteActionRequest/gap-pr:Action"
       message-name="gap:ExecuteActionRequest" message-part="ExecuteActionRequest"/>
  <message-construction io-ref="gap-pr:ExecuteActionResponse"
       message-name="gap:ExecuteActionResponse" message-part="ExecuteActionResponse"/>

  <map action="destroy" operation="wsrf-rlw:Destroy"/>
  <message-construction io-ref="gwsrf-rl:DestroyRequest"
       message-name="wsrf-rlw:DestroyRequest" message-part="DestroyRequest"/>
  <message-construction io-ref="wsrf-rl:DestroyResponse"
       message-name="wsrf-rlw:DestroyResponse" message-part="DestroyResponse"/>
 </service-binding>

 <service-metadata>
  <wsdl-type message-name="gap:ExecuteActionRequest"
     element-name="gap-pr:ExecuteActionRequest"/>
 </service-metadata>

 <broker-interface>
  <service-URI>
    http://dragonfly.risc.uni-linz.ac.at:8080/wsrf/services/GapPort
  </service-URI>
  <broker-interface-description>
    http://dragonfly.risc.uni-linz.ac.at:8080/wsrf/services/GapPort?wsdl
  </broker-interface-description>
 </broker-interface>

</service>

</definitions>
```

References

[1] Roberto Chinnici et al, Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3 Consortium, December 2005. http://www.w3.org/TR/wsdl20/

[2] Martin Gudgin et al, SOAP Version 1.2 Part 1: Messaging Framework, W3 Consortium, June 2003. http://www.w3.org/TR/soap12-part1

[3] The OASIS Web Services Resource Framework, December 2005. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[4] Tim Banks, Web Services Resource Framework (WSRF) - Primer, December 2005. http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-01.pdf

[5] Steve Graham et al, Web Services Resource 1.2 (WS-Resource), December 2005. http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-cs-01.pdf

[6] Don Box et al, Web Services Addressing (WS-Addressing), August 2004. http://www.w3.org/Submission/ws-addressing/

[7] Technology report, Web Services Addressing (WS-Addressing), December 2005. http://xml.coverpages.org/ws-Addressing.html

[8] GAP - Groups, Algorithms, Programming - a System for Computational Discrete Algebra, December 2005. http://www-groups.dcs.st-and.ac.uk/ gap/

[9] Apache WebServices - Apache WSRF, October 2005. http://ws.apache.org/wsrf/

[10] Steve Graham et al, Web Services Resource Properties 1.2 (WS-ResourceProperties), April 2005. http://docs.oasis-open.org/wsrf/2005/03/wsrf-WS-ResourceProperties-1.2-draft-06.pdf

[11] Latha Srinivasan et al, Web Services Resource Lifetime 1.2 (WS-ResourceLifetime), March 2005. http://docs.oasis-open.org/wsrf/2005/03/wsrf-WS-ResourceLifetime-1.2-draft-05.pdf

[12] Tom Maguire et al, Web Services Service Group 1.2 (WS-ServiceGroup), June 2004, http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-02.pdf

[13] Steve Tuecke et al, Web Services Base Faults 1.2 (WS-BaseFaults), April 2004. http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-02.pdf

[14] OASIS Web Services Business Process Execution Language (project website), December 2005. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[15] Alexandre Alves et al, Web Services Business Process Execution Language Version 2.0, Commitee draft, January 23, 2006. http://www.oasis-open.org/committees/download.php/16525/wsbpel-specification-draft%20feb%2001%202006%20no%20tracking.htm

[16] The Twister Engine (project website), December 2005. http://www.smartcomps.org/twister/

[17] James Clark et al, XML Path Language (XPath) Version 1.0. November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116

[18] Twister (project website), December 2005. http://www.smartcomps.org/twister/

[19] A Framework for Brokering Distributed Mathematical Services. Reseach Institute for Symbolic Computation (RISC), April 2004. http://www.risc.uni-linz.ac.at/projects/basic/mathbroker.

[20] OpenMath (project website), December 2005. http://www.openmath.org/

[21] MONET — Mathematics on the Web, MONET Consortium, April 2004. http://monet.nag.co.uk

[22] Mike Dewar, The MONET Ontologies in OWL, In MONET Workshop, Bath, UK, March 2004. http://monet.nag.co.uk/cocoon/monet/MONETWorkshop.html

[23] Olga Caprotti and Wolfgang Schreiner. Towards a Mathematical Service Description Language. In International Congress of Mathematical Software ICMS 2002, Bejing, China, August 17–19, 2002. World Scientific Publishing, Singapore.

[24] Mathematical Services Description Language (MSDL), Reseach Institute for Symbolic Computation (RISC), April 2004. http://poseidon.risc.uni-linz.ac.at:8080/mathbroker/results/xsd.html.

[25] Mathematical Service Description Language: Final Version. The MONET Consortium (IST-2001-34145). Deliverable D14. http://monet.nag.co.uk/cocoon/monet/publicdocs/monet-msdl-final.pdf

[26] ebXML, http://www.ebxml.org/

[27] Mike Dewar, Identifying and Brokering Mathematical Web Services, The Web Services Journal, 3(8), August 2003. http://www.syscon.com/webservices

[28] Olga Caprotti. Extending MONET to the MathBroker Information Model. Project report, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, June 2003.

[29] Rebhi Baraka, A Framework for the Registration and Discovery of Mathematical Services, Ph.D. thesis in progress (completion expected in 2006).

[30] OWL-S 1.1 Release, November 2004. http://www.daml.org/services/owl-s/1.1/

[31] WSMO - Web Service Modeling Ontology, December 2005. http://www.wsmo.org/

[32] XMLBeans (project website), December 2005. http://xmlbeans.apache.org/

[33] Velocity Engine (project website), December 2005. http://jakarta.apache.org/velocity/