Extending Web Service Technology towards an Earth Observation Integration Framework

Marcello Mariucci[1,2,*] and Bernhard Mitschang[2]

[1] European Space Research Institute, European Space Agency,
00040 Frascati (RM), Italy
[2] Institute of Parallel and Distributed Systems, University of Stuttgart,
70569 Stuttgart, Germany
{mariucci, mitsch}@informatik.uni-stuttgart.de

Abstract. In this paper we describe the implementation of a service-based application integration solution for the complex domain of Earth Observation (EO) application systems. The presented approach is based on an EO integration framework. It supports the concatenation of disparate software applications to flexible EO process chains. Resulting EO services are provided to end users by means of Web Service technology. We demonstrate that current standard technology is not sufficient to dynamically publish and interactively invoke EO services over the Web. We describe necessary extensions and adaptations.

1 Introduction

An Earth Observation (EO) integration framework is a service-based application integration approach for the interdisciplinary domain of EO application systems [1]. It facilitates the development and execution of EO services by supporting the integration of disparate applications to EO process chains. EO services are based on the intensive use of large data sets from space. They process raw EO data sets into increasingly specialized products until a certain level of quality is achieved. This processing requires the tight cooperation of several distributed experts, and intensive computation across a coordinated sequence of both, interactive and automatic processing steps. Appropriate examples for such EO services are the generation of weather forecasts, forest fire detection, and oil slick monitoring. Fig. 1 illustrates a simplified process flow of a representative EO service. It depicts the generation and use of forest fire observation products.

Radar images and meteorological vectors are regularly sensed and acquired by related acquisition stations ①. *Data Providers* monitor the creation of these EO data sets, and ensure their correct ingestion into the Central Server ②. The Central Server pre-processes the received data for fire detection purposes, and forwards them to an

[1] Work was done while the author was visiting the European Space Agency (ESA/ESRIN).

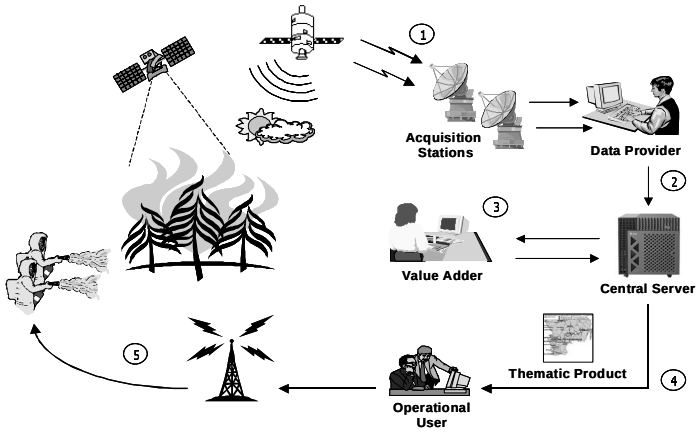

Fig. 1. Oversimplified EO Service for Forest Fire Detection

appropriate EO imagery expert. This so-called *Value Adder* analyzes, filters, maps, and merges the processed data, and eventually extracts fire observation parameters like burned area contour and perimeter ③. Analysis results are then sent back to the Central Server where they are packaged into thematic products. Generated thematic products are sent to *Operational Users* ④, who assess and validate them by using locally available information ⑤. Based on these results, they can initiate and coordinate operative actions, such as fire-fighting and rescue operations ⑥.

The example scenario shows that the seamless processing of EO services requires the integration of heterogeneous tools and management systems. That is, transitions between processing steps have to be automated and coordinated. Furthermore, data set representations have to be based on a common model. The European Space Agency has approached with mainly two research directions the implementation of such an integration solution. In the following Section 2 we briefly outline and analyze these initiatives. In Section 3 we present our solution of an EO integration framework. We introduce the service-based application integration approach, and highlight the related integration model. In Section 4 we describe the realization of our integration framework. We emphasize the use of Web service technology, and present necessary extensions and adaptations to dynamically publish and interactively invoke EO services over the Web. Section 5 summarizes and concludes the paper with an outlook to further work.

2 Related Work

The European Space Agency (ESA) has approached with mainly two activity directions the development of an EO integration framework. On the one side it has pursued a *top-down approach*, which has analyzed the extension of an existing EO application infrastructure towards a generic, multi-application platform. On the other side, it has investigated a *bottom-up approach*, which has developed an interoperable

protocol environment for coupling interdisciplinary EO facilities to integrated solutions.

ESA's top-down initiative is introduced and analyzed in [1]. The basic idea of this approach is to reuse generic functions of an already implemented EO application systems, and to build up a common framework for EO application systems. The architecture is based on a modular, object-oriented CORBA design, and deploys function modules on geographically distributed computing machines. EO services are created by combining appropriate modules to thematic process chains. Missing modules are added to the framework on demand. Corresponding process descriptions are hard-coded within so-called server modules, and provided as EO services via CORBA protocols. Analyses of this approach turned out that the infrastructure is extremely inflexible and not particularly suitable to form an EO integration framework. The creation of EO services represents a huge overhead, leading to twisted and hardly manageable software systems. Furthermore, appropriate process control and metadata management facilities are missing [1].

ESA's bottom-up approach is related to the development of a Multiple Application Support Service (MASS) protocol environment [2]. It defines an interoperable infrastructure that aims at coupling clients, tools, and management systems. Basically, MASS protocols extend the functionality of CORBA protocols and services for the purpose of EO application systems. Analyses concerning the suitability of this approach to form an EO integration framework revealed performance and flexibility problems. Furthermore, they emphasized that the MASS infrastructure is lacking of an information base and development environment for managing system resources and structures. The infrastructure does not provide a comprehensive overview of software elements available in the system. MASS information bases are restricted to interface specifications. Further information like architectural design of the system or deployment information of software elements are either hard-coded in the single software products or written in some documents. This makes it difficult for EO service developers to choose the right components during EO service creation and extension.

Based on experiences gained in assessing ESA's approaches, we designed and implemented an alternative solution for an EO integration framework. The following section briefly introduces our service-based integration approach.

3 Service-based Application Integration Approach

Our EO integration framework provides an open software infrastructure for supporting the inherent complexity of EO application systems. It facilitates the coordinated integration of disparate tools and management systems along EO process chains. In addition, it exposes related EO services over the Web by applying Web service technology. As depicted in Fig. 2, EO services are published and registered in a UDDI service directory ①. End users can discover EO services they are interested in by enquiring the UDDI ②. The information they retrieve from the directory suffices to localize and use the EO service ③. For each EO service a related process flow de-

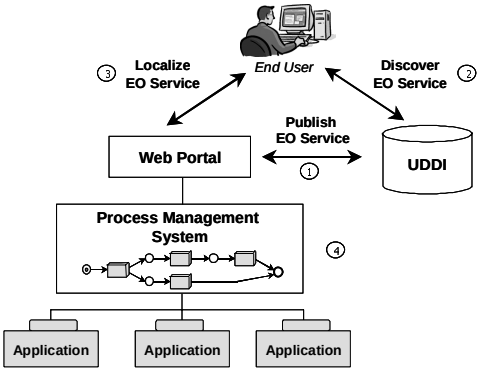

Fig. 2. Service-based EO Integration Framework

scription is available. It is used by a process management system to execute the request EO service, and to coordinate related processing steps ④.

Our service-based application integration approach is divided into three phases, i.e. design time, development time, and run time (see Fig. 3). The core element of the integration process is a shared integration model. It structures metadata about any resource, activity, and process of the framework, and represents the source from which documentation and code generation can be directly derived. The model is based on UML, and intended for process-oriented application integration purposes within EO integration frameworks. The main value of the model-driven EO integration framework is its flexibility and technological independency. By truly decoupling software specifications from implementation details, heterogeneous realizations of singe software products are abstracted. Thus, software products are integrated into flexible EO process chains, which can be dynamically reconfigured and customized as the environment changes.

At design time the focus is on the creation of the integration model. It is designed by integration architects once at the beginning of the integration process. The model specifies the structure of integration artifacts for the persistent storage of related instantiations. Development begins after the integration model is made available. Integration developers use client tools to collaboratively instantiate the integration model. Applications are thus registered, linked to their respective implementations, and assembled to combined EO services. The integration model is then used to generate appropriate code for EO service publication and execution. At run time, end users employ browser tools and proprietary clients to discover, locate, and invoke EO services. Requests are forwarded to the execution engine, which orchestrates the corresponding EO service processing.

4 Prototype Implementation

Based on the presented service-based integration approach, we implemented an EO integration framework as part of the ARSENAL project [3]. The project aimed at the

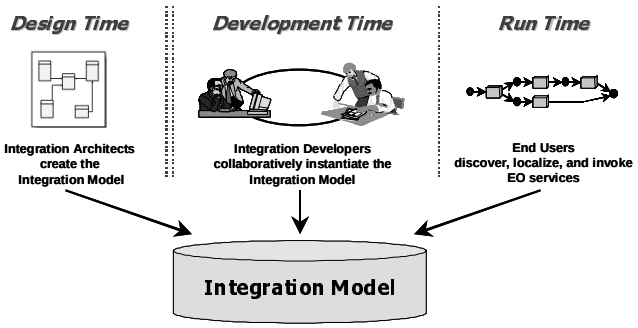

Fig. 3. Application Integration Process

design and implementation of a framework infrastructure for the flexible integration of applications within the EO domain. The main requirements were the dynamic reuse of application functions, the life cycle management of EO service structures, and the ubiquitous access to and controlled execution of EO service instances. Furthermore, the infrastructure had to be built upon commercial products, in order to guarantee best possible system stability, reliability, and low development costs. In this section, we evaluate middleware technologies regarding their use within our framework prototype. Then, we describe each integration phase in detail, and evaluate commercial middleware technologies regarding their use within our framework prototype. We emphasize selected implementation issues required for the seamless combination of those middleware products.

4.1 Middleware Technology Analysis

Metadata created during the integration process needs to be consistently stored and managed during the entire software life cycle. Its right treatment seriously enhances the integration process in terms of dynamics, flexibility, and adaptability to the constantly changing business environment. *Repository* is a technology that deals with the whole spectrum of metadata management. It provides a centralized, persistent storage, helps in reducing redundancies and inconsistencies, and improves reuse. Besides basic database management functions, the repository provides functions for the seamless management of metadata throughout the entire software life cycle (e.g. versioning, configurations). Furthermore, it offers indispensable functions for the collaborative management of system metadata (e.g. workspaces, contexts). The main value of the repository technology, and its functions are summarized in [4].

EO services describe process flows between applications within the EO domain. Different forms of middleware have been introduced by the computing community to enable integration and automation of processes. In general, a *Process Management System (PMS)* provides a central point of control for defining process flows and orchestrating their execution. It records the execution state of the process, and routes requests to applications to execute tasks [5]. In order to unify the access to applications and EO services standard interface definitions are required. Finally, *Web Ser-*

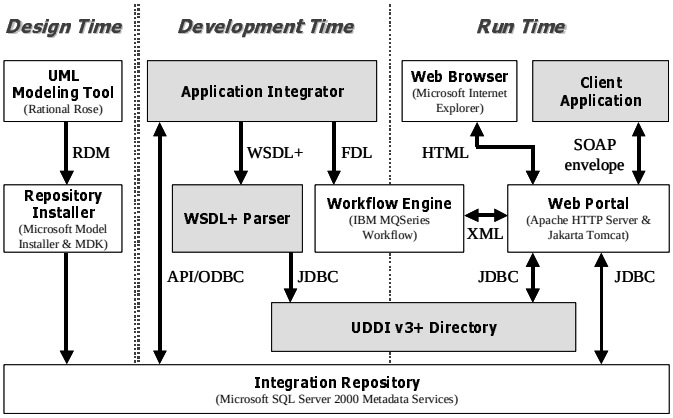

Fig. 4. ARSENAL Implementation Architecture

vices define techniques for describing, discovering, and locating software components on a global network. These techniques are programming language- and protocol-neutral, and provide ubiquitous and transparent access to system resources [6].

The introduced middleware technologies basically meet the requirements on our EO integration framework infrastructure. In the following sections we discuss the use of related commercial technology products in the EO integration framework. We emphasize additional 'glue' components required for coupling these tools to seamlessly support EO application systems. Fig. 4 illustrates the ARSENAL implementation architecture. White boxes identify Commercial Off-The-Shelf (COTS) tools, whereas grey boxes identify 'glue' components. The figure represents the big picture for the remainder of this chapter.

4.2 Design Time Support

Our EO framework prototype is based on the object-oriented repository of Microsoft's SQL Server 2000 Meta Data Services (see Fig. 4). The repository provides a powerful API for the information model definition and management during the entire software life cycle. In addition, it supplies a Model Installer, and a Model Development Kit (MDK) for the Rational Rose Modeling Tool. Both tools facilitate the installation of UML models into the repository. By means of these features, integration architects initially define the integration model by using UML, export it as a Repository Distributable Model (RDM) file, and install it in the integration repository. This procedure is done once at the beginning of the integration process.

4.3 Development Time Support

At development time, integration developers are supported to collaboratively instantiate the integration model. The Application Integrator tool (see Fig. 4) supports

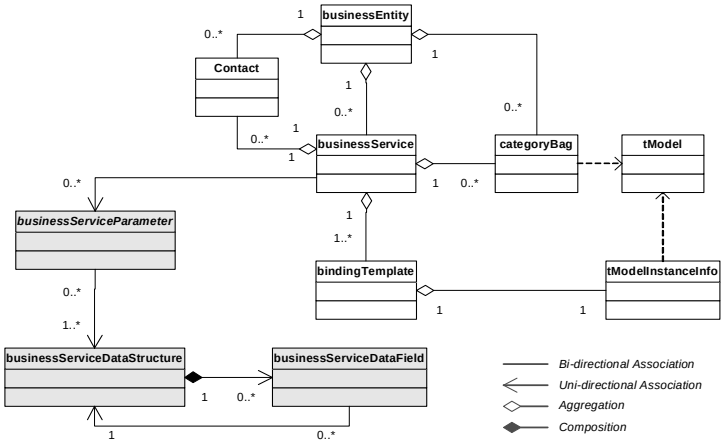

Fig. 5. UDDI v3+ Data Model

integration developers to collaboratively create EO services. It provides a user-friendly GUI, and uses the repository API to create, navigate, and manipulate the integration model content, basically by inserting, deleting, and updating object instances. In addition, it enables asynchronous collaborative work by employing Microsoft's repository features, such as version, configuration, and workspace management. The Application Integrator tool also supports the generation of appropriate, technology-specific code for publishing and executing EO services. For EO service execution purposes, it creates a Flow Definition Language (FDL) file, and exports it to the workflow engine. For EO service publishing purposes, it generates an extended Web Service Definition Language (WSDL+) file. This WSDL+ file is then parsed and imported into the UDDI v3+ registry. Extensions regarding the WSDL and the UDDI specifications are required for the interactive Web service invocation, and for the automatic registration of EO services into the UDDI registry. They are described in detail in the following subsections.

UDDI Extensions for Interactive Web Service Invocation. EO services interfaces are described as Web services and published in a private UDDI registry. This UDDI registry is a database infrastructure that enables end users and their client applications to quickly and easily find EO services [7]. It links each EO service description to its corresponding WSDL interface definition file, which abstractly describes related access structures. By means of this file, end users can integrate the EO service within their application code, and locate and invoke it through remote procedure calls.

Besides this standard form of invoking EO services, the ARSENAL prototype facilitates the interactive invocation of EO services from within a Web browser tool. That is, end users are not forced to retrieve the corresponding WSDL interface definition file to access the EO service, but are enabled to directly and interactively invoke the EO service from within their standard Web browsers.

Two approaches were analyzed to implement this feature within the ARSENAL prototype. The first approach was related to the enhancement of the Web portal to process WSDL interface definition files. As soon as an end user requests an EO service invocation, the Web portal parses the corresponding WSDL file, and dynamically generates a Web page requesting the EO service input parameters. It then sends the request together with its parameters to the corresponding port to automatically launch the EO service execution. However, this solution resulted to be too time-consuming, since the WSDL file is processed and parsed at run time. Consequently, the second approach regarded the extension of the UDDI registry to additionally manage WSDL interface definitions. That is, the UDDI registry is enhanced to enable the storage and retrieval of EO service access structures. In this way, WSDL files can be parsed at development time, and EO service interface definitions directly used to dynamically generate a Web page requesting the EO service input parameters. Although this approach extends the UDDI standard in a proprietary fashion, the performance and response time of the Web portal is enormously enhanced.

The ARSENAL prototype implements the second approach, and, therefore, extends the UDDI registry with EO service access structures. Fig. 5 illustrates the enhanced UDDI data model. It refers to the UDDI v3 standard and extends it by three entities, namely:

- **businessServiceParameter** for the specification of the EO service parameters,
- **businessServiceDataStructure** for the specification of EO service parameter data types, and
- **businessServiceDataField** for the specification of complex data types.

These three additional entities are highlighted in Fig. 5. The resulting UDDI registry is called UDDI v3+ registry.

Automatic Registration of EO Services into UDDI v3+. Our prototype implementation provides the feature to automatically generate both, WSDL interface and WSDL implementation files for EO services. Based on EO service descriptions stored in the repository, appropriate WSDL files are generated, parsed, and registered into the UDDI v3+ registry. Usually, the registration of Web services in a UDDI registry is performed in interactive mode, i.e. the service provider connects to the UDDI registry via a Web browser, registers its business entity, and describes all provided Web services. Since our repository includes all required information, we additionally support automatic registration of EO services in the UDDI v3+ registry. For this purpose, we had to appropriately extend the WSDL implementation file by 'business entity' and 'category bag' specifications. Fig. 6 lists a fragment of an exemplary WSDL+ implementation file. Extensions are highlighted.

The WSDL+ implementation file is sent to the WSDL+ parser. The WSDL+ parser is based on Apache's Xerces DOM parser, and verifies the syntactical correctness of the WSDL+ file. In addition, it inserts related entries into the UDDI v3+ registry.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions ...>
  <documentation> ... </documentation>
  <import ... />
  <businessEntity>
    <name>Space Data Technology, Inc.</name>
    <description>Satellite Data Provider</description>
    <contacts>
      <contact useType="Organization Manager">
        <personName>Marcello Mariucci</personName>
        <details>mariucci@spacedatatech.com</details>
      </contact>
    </contacts>
    <categoryBag>
      <keyedReference tModelKey="UUDI:ClAD242-9343-2321" keyName="Earth Science" keyValue="Data"/>
      <keyedReference tModelKey="UUDI:D4EG343-3534-2316" keyName="Data" keyValue="Provider"/>
    </categoryBag>
  </businessEntity>
  <service ...>
    <documentation> ... </documentation>
    <categoryBag>
      <keyedReference tModelKey="UUDI:D3ER342-2345-6545" keyName="Data" keyValue="Ingestion"/>
    </categoryBag>
    <port ... > ... </port>
  </service>
</definitions>
```

Fig. 6. WSDL+ Implementation File

Generation of a Business Process Definition File. The Application Integrator tool is capable of exporting workflow specifications of EO services to a FDL file. This file includes instructions for the workflow engine to properly execute EO services at run time. To prepare EO service executions, such a file is generated and imported into the respective BPMS. FDL is a proprietary workflow specification format of IBM. However, the Application Integrator tool is open and flexible enough to adapt its code generation procedure to other workflow description formats, such as the emerging BPEL [8] format.

4.4 Run Time Support

At run time, end users discover, locate, and invoke EO services by applying Web service technology. By means of a Web browser, end users query the UDDI v3+ registry to retrieve information about available EO services and their access locations. More specifically, the Web browser connects to the Web portal and requests an appropriate Java Server Page (JSP). The request is forwarded to the Servlet engine which executes corresponding processing steps, queries the UDDI v3+ registry, and sends related information back to the end user via the Web portal. The UDDI v3+ registry provides descriptive information about EO service interfaces and access points. End users use this data to locate EO services, and to invoke them by specifying related parameters.

EO services are offered through the SOAP (Simple Object Access Protocol) [9]. SOAP messages for the invocation of EO services are either created by client applications, or automatically generated by the Web portal for interactive EO service invocations. They are then sent to the corresponding access point by using the HTTP post protocol. Through this access point the message is forwarded to the SOAP adapter, which processes it, and sends corresponding execution instructions to the

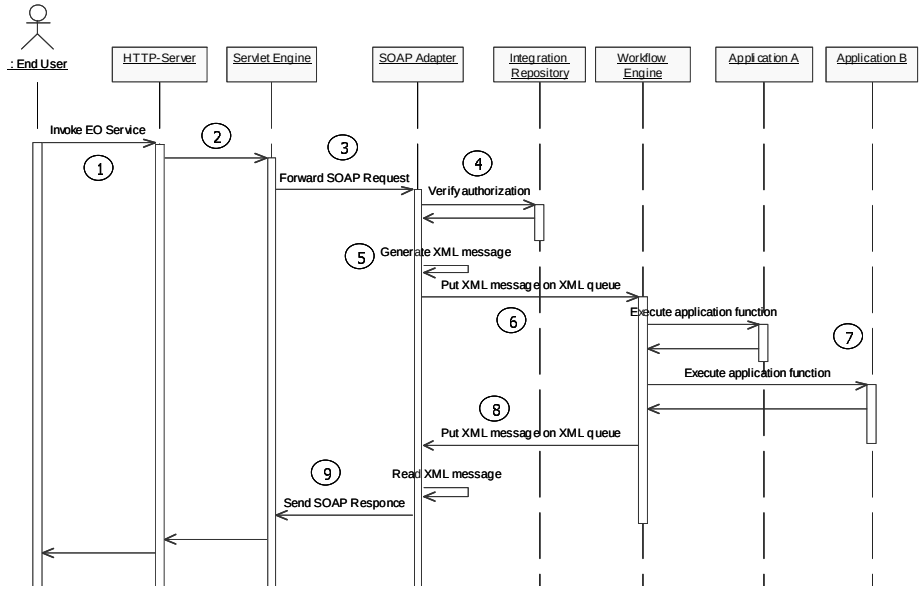

Fig. 7. UML Sequence Diagram for EO Service Invocation Handling

workflow engine. Execution instructions are coded in the Extensible Markup Language (XML) format [10]. The workflow engine provides a reliable and robust infrastructure for the orchestration of EO services related process chain. Results are sent back to the Web portal, which either forwards them to the client application, or sends them via email to the end user. In the reminder of this section, we briefly describe the dynamic invocation of EO services via SOAP messages.

Dynamic SOAP Adapter for a Workflow Management System. As described above, end users can send SOAP request messages to an access point in order to initialize EO service executions. For that purpose, the Web portal integrates a dynamic SOAP adapter, which is able to parse SOAP messages and to forward related execution instructions to the workflow engine. Our SOAP adapter is based on Apache SOAP, which per se provides a SOAP handler to parse SOAP messages and to map requests to static Java classes. Unfortunately, this implementation approach does not meet our demands, as in our integration infrastructure Web services are dynamically created, and, thus, Web service related Java classes do not exist in advance. A new dynamic SOAP handler, which additionally adapts to a workflow management system, was implemented. Fig. 7 depicts a UML sequence diagram regarding the invocation of EO services via SOAP in our EO integration framework.

The end user sends a SOAP message to the HTTP Server ①, which forwards the requests via Servlet engine ② to the SOAP adapter ③. The SOAP adapter verifies the request. In order to check whether the service parameters are correct and whether the client is authorized to execute the EO service, the adapter interacts with the integration repository ④. This connection is also used to get service execution instructions. The SOAP adapter generates XML statements about these execution instruc-

tions along with the service parameters ⑤, and sends them via XML queues to the workflow engine ⑥. The workflow engine starts the EO service among the involved applications ⑦. At the end, it sends a status report back to the SOAP adapter ⑧, which forwards it to the end user via SOAP ⑨.

5 Conclusions and Future Work

In this paper we presented a flexible EO integration framework for supporting the inherent complexity of EO application systems. We focused on a service-based application integration approach and its realization. We emphasized our prototype implementation, which integrates and adapts COTS tools for workflow management, repository, and Web services technology. The prototype infrastructure has been successfully applied in real EO environments, which proves that flexible application integration could be achieved by the right composition and adaptation of existing technology. Up to now, the EO integration framework infrastructure is limited on a few scientific applications and a selective user community. However, visions concern the use of such frameworks to foster commercial EO applications, and to build up a market for EO data and application providers.

In our approach, Web service technology is not used to link applications inside the EO integration framework. Since Web service technology products are still in an experimental fashion, we preferred to use more expensive, but proven and reliable, products. Future work regards the employment of Web service technology for integrating internal key systems. Further future work is concerned to emerging Grid technologies [11]. As of now, Grid technology is perceived as an orthogonal issue. Coupling Grid with our EO integration framework offers the possibility of transparently executing application functions on best possible resource allocation configurations. Furthermore, security solutions of local applications could be maintained, and intensive computation applications executed on dynamic high performance resources. The employment of Grid would greatly simplify the sharing and dissemination of applications, and enhance the quality, effectiveness and efficiency of our application integration approach.

References

1. Mariucci, M., Mitschang, B.: On Making RAMSES an Earth Observation Application Framework. In: Smari, W.W., Melab, N., Chen, S.-C. (eds.): The 2nd International Conference on Information Systems and Engineering. The Society for Modeling and Simulation International, Vol. 34, Nr. 2. San Diego (2002) 67–72.
2. Doherty, C., Usländer, T., Landgraf. G.: Multiple Application Support Services. An ESA Protocol for EO Application Clients. In: Earth Observation & Geo-Spatial Web and Internet Workshop. Committee on Earth Observation Satellites. London (2000). http://webtech.jrc.it.

3. European Space Agency, University of Stuttgart: ARSENAL Project. Official Homepage. http://www.informatik.uni-stuttgart.de/ipvs/as/projekte/arsenal/index_engl.html.
4. Bernstein, P.A., Dayal, U.: An Overview of Repository Technology. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.): 20th International Conference on Very Large Data Bases. Santiago (1994) 705-713.
5. Dayal, U., Hsu, M., Ladin, R.: Business Process Coordination: State of the Art, Trends, and Open Issues. In: Apers, P.M.G., Atzeni, P., Ceri, S., Paraboschi, S., Ramamohanarao, K., Snodgrass, R. (eds.): 27th International Conference on Very Large Data Bases. Rome (2001) 3-13.
6. Leymann, F.: Web Services: Distributed Applications without Limits – An Outline. In: Weikum, G., Schöning, H., Rahm, E. (eds.): 10th BTW 2003 Datenbanksysteme für Business, Technologie und Web. Leipzig (2003) 2-23.
7. Belwood, T., et al.: UDDI 3.0. UDDI Spec Technical Committee Specification. http://www.uddi.org.
8. Leymann, F., Roller, D.: Business processes in a Web service world. A quick overview of BPEL4WS. IBM report. http://www-106.ibm.com.
9. World Wide Web consortium (W3C): Simple Object Access Protocol (SOAP) 1.1. W3C Note 08 May 2000. http://www.w3.org/TR/SOAP
10. World Wide Web consortium (W3C): Extensible Markup Language (XML). http://www.w3c.org/XML/.
11. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann (1999).