ARSENAL: Model-driven Earth Observation Integration Framework

Marcello Mariucci Frank Wagner Gunter Martens Jens Künzl
(University of Stuttgart, Germany) [1]

An Earth Observation (EO) integration framework is an application integration solution for supporting the development and execution of EO services. 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. EO services can be characterized as highly flexible structures, which constantly need to be adapted to evolving spacecraft and processing technologies. We suggest a model-driven EO integration framework solution that adequately copes with the flexible development, customization, and execution of reusable EO services. Our prototype includes a comprehensive integration model that accurately handles system metadata throughout the software life cycle, and significantly enhances the EO service development process in terms of quality, reuse, and adaptability. The prototype employs repository technology for managing model related issues, as well as workflow and Web service technology for execution purposes. It is mainly built upon commercial products, which are seamlessly combined by appropriate 'glue' components.

1. Introduction

Earth Observation (EO) application systems are characterized by the provision of thematic, value-adding services, which are based on the intensive use of EO data from space. An appropriate example is the RAMSES system. RAMSES is an operational, Web-based EO application system that provides oil spill detection and monitoring services based on satellite technology [1]. Figure 1 illustrates a simplified RAMSES process flow.

Radar images of the sea surface and related meteorological vectors are regularly acquired ① and sent to the RAMSES server ②. The RAMSES server pre-processes the received EO data for oil slick detection purposes, and forwards them to an appropriate EO imagery expert. This so-called value adder analyzes the processed data to detect, and eventually extracts oil slick parameters like contour, perimeter, and development forecasts ③. Analysis results are then sent back to the RAMSES server, which generates an appropriate status report. In addition, if the value adder detected an oil slick, it creates related oil slick observation products. Reports and observation products are then sent to operational users ④, who assess and validate them by using locally available tools and data ⑤. Based on these results, users might decide to take operative actions, such as cleaning operations, or sending out airplanes to trap polluters ⑥. Finally, the operational users send feedback and eventual on-site reports to the RAMSES server for statistical reasons ⑦.

[1] Work was done while the first author was visiting the European Space Research Institute (ESA/ESRIN)

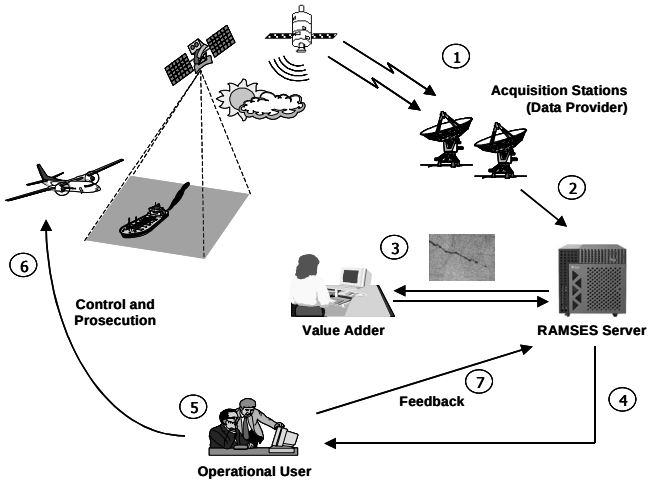

Figure 1: Oversimplified EO Service for the Detection and Prevention of Forest Fires

The RAMSES example shows that the processing of EO data requires a coordinated sequence of both, interactive and automatic processing steps, and involves several organizations. These entities are typically geographically distributed, focused on well-defined core competences, and independently organized with their own application systems. By assembling these application functions, value-adding *EO services* are created, published, and marketed. EO application systems are currently made up of various isolated tools and management systems, which offer insufficient interoperability. These software products optimally support experts in performing their specific tasks, however, they hardly provide any facilities to interoperate with other tools. The purpose of an *EO integration framework* is to provide a reusable software infrastructure for supporting the inherent complexity of EO application systems. It assists users to collaboratively create, process, and monitor EO services during the entire software life cycle. In addition, it facilitates the flexible integration of disparate tools and management systems along related EO process chains. Our demo presents an implementation of an operational EO integration framework prototype.

In [2] we analyzed requirements on an EO integration framework, and identified several characteristics that are crucial for the seamless management of EO services. The main issue concerns the flexible support of collaborative, on-demand changes and extensions to EO services. Our prototype implements a highly flexible EO integration framework, which enables collaborative adaptation of EO services to the continuously improving spacecraft and data processing technologies. Furthermore, it allows the use of EO services in an experimental fashion, i.e. EO services can be initially built, tested, and incrementally enhanced. This evolution cycle can be iterated until a certain level of quality is achieved. To provide the required flexibility and manageability, our prototype is based on an *integration model*. It structures metadata about any resources, activity, and process of the framework, and represents the source from which documentation and code generation can be directly derived. In addition, it accurately treats metadata throughout the software life cycle, which significantly enhances the integration process in terms of quality, flexibility, and adaptability to the constantly changing situation.

The integration model represents the core element of our application integration approach. Its main idea is to describe the structure of applications, and to express how their functions are combined via control and data connectors to constitute EO services. In this way, the sequence of interoperation between applications can be defined, and the related data exchange specified. Furthermore, the integration model serves as template from which EO services are instantiated through the Web, i.e. EO service interface descriptions and workflow specifications are generated. The integration model

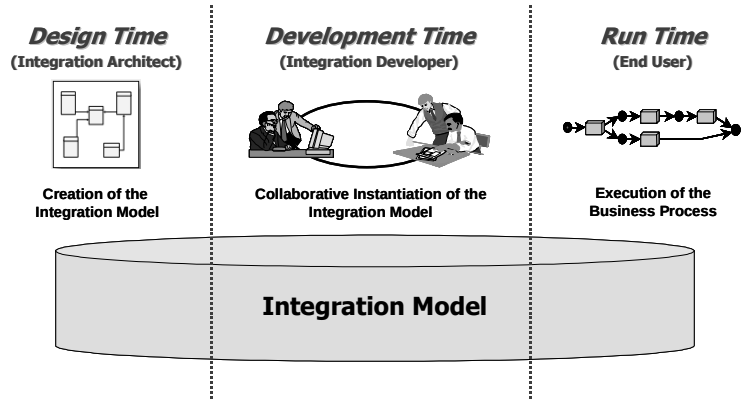

Figure 2: Model-driven Application Integration Process

metadata is continuously used during the entire integration process. As depicted in Figure 2, our model-driven integration process is divided into three phases, i.e. design time, development time, and run time. At *design time* the focus is on the creation of the integration model. It is designed by *integration architects*, who use artefacts tools to specify the structure of integration artefacts. The model management has to provide for persistent storage and continuous metadata support. *Development* begins after the integration model is made available. *Integration developers* use 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 invocation and execution. At *run time*, *end users* employ browser and proprietary client tools to discover, locate, and invoke EO services through the Web. Requests are forwarded to the execution engine, which orchestrate corresponding EO service processing.

2. Architecture and Technology

In this section, we present the building blocks of our prototype and show how they are used and combined to support users during the entire integration process.

2.1. Design Time Support

Our EO framework prototype is based on the object-oriented repository of Microsoft's SQL Server 2000 Meta Data Services. 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, the integration architect initially defines the integration model using UML, exports it as a Repository Distributable Model (RDM) file, and installs it in the integration repository (see Figure 3).

2.2. Development Time Support

At development time, integration developers are supported to collaboratively instantiate the integration model. Figure 4 depicts the prototype architecture during development time. White boxes of the implementation architecture identify Commercial Off-The-Shelf (COTS) tools. Grey boxes represent 'glue' components required for coupling those tools and providing specific EO framework functionality. The Application Integrator supports integration developers to collaboratively manage EO

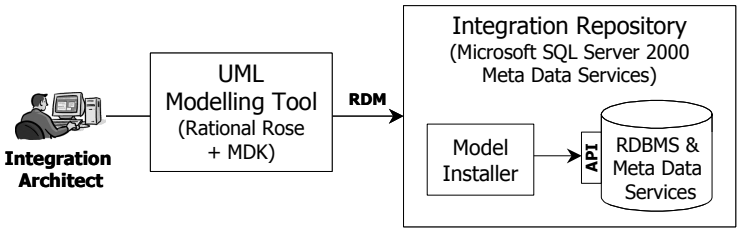

Figure 3: Prototype for Design Time Support

framework metadata ①. It provides a user-friendly GUI, and uses the repository API to create, navigate and manipulate the integration model content, basically by insertion, deletion and updating object instances ②. In addition, it enables asynchronous collaborative work by employing Microsoft's repository features such as version, configuration and workspace management. The Application Integrator also supports the generation of appropriate, technology-specific code for publishing and executing EO services. In order to prepare for EO service execution, an appropriate Business Process Definition (BPD) file is generated ③ and exported to the BPMS ④. However, in order to publish designed EO services, related interface specifications are exported to an extended Web Service Definition Language (WSDL+) file ⑤. This WSDL+ file is then parsed and imported into the UDDI v3+ registry ⑥ in order to publish the EO service. Extensions regarding WSDL and UDDI specifications are required for the interactive Web service invocation, and for the automatic registration of EO services into the UDDI registry.

2.3. Run Time Support

At run time, end users discover, locate, and invoke EO services by applying Web service technology. Figure 5 depicts the prototype architecture for run time usage. Again, white boxes identify COTS tools, and grey boxes represent 'glue' components. By means of a Web browser, end users query the UDDI v3+ registry to retrieve information about available EO services and their access locations. In detail, the Web browser connects to the HTTP server and requests an appropriate Java Server Page (JSP) ①. The request is forwarded to the Servlet engine ②, which executes corresponding processing steps, queries the UDDI v3+ registry, and sends related information back to the end

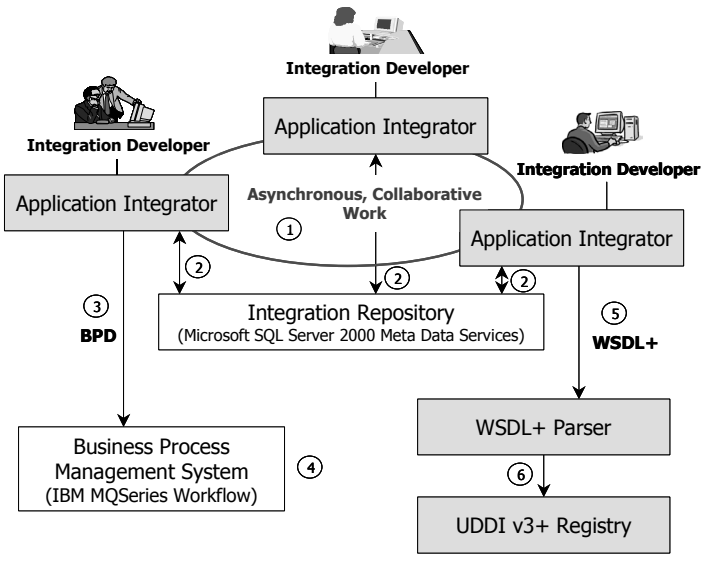

Figure 4: Prototype for Development Time Support

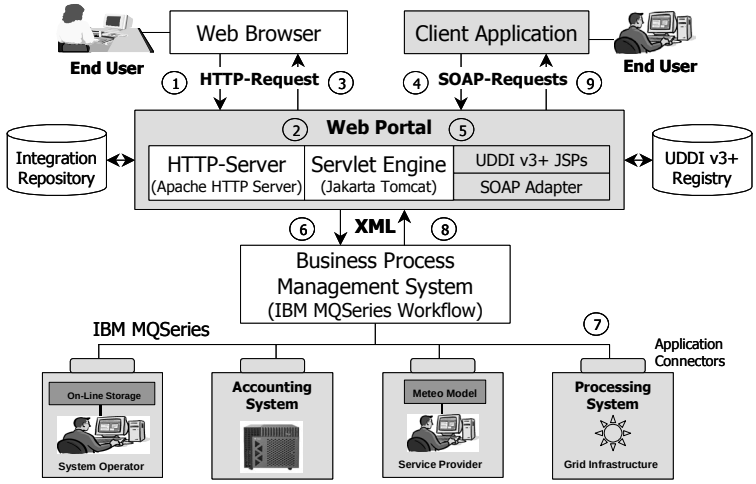

Figure 5: Prototype for Run Time Support

user via the HTTP server ③. The UDDI v3+ registry includes 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 provided through SOAP (Simple Object Access Protocol). SOAP messages for the invocation of EO services are created by client applications, and sent to the appropriate access point by using the HTTP post protocol ④. Through this access point the message is forwarded to the SOAP adapter ⑤, which processes it and sends corresponding execution instructions to the BPMS via XML ⑥. Our BPMS is implemented by IBM's MQSeries Workflow Management System, which provides a reliable and robust infrastructure for the orchestration of EO services and its application function interrelations ⑦. Results are sent back to the Web portal ⑧, which are then forwarded to the client application ⑨.

3. Contribution

Our prototype is a flexible EO integration framework that controls the inherent complexity of integrating applications within the EO domain. We focused on the management of a metadata model and its role during the application integration process. Our prototype implementation integrates and adapts COTS tools for workflow management, repository, and Web service implementations. The prototype infrastructure has been successfully applied in real EO environment test cases, which proves that flexible application integration can be achieved by the right composition and adaptation of existing technology. Up to now, the EO integration framework infrastructure is limited to 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.

References

[1] MARIUCCI M.; et al. 2000. "RAMSES: An Operational Thematic EO-Application on Oil Spill Monitoring. System description". In *Proceedings of the Earth Observations & Geo-Spatial Web and Internet Workshop 2000 Conference* (London, UK, April 17-19). EOGEO 2000.

[2] MARIUCCI M.; B. Mitschang. 2002. "On Making RAMSES an Earth Observation Application Framework". In *Proceedings of the 2nd International Conference on Information Systems and hEngineering* (San Diego, USA, July 14-18). ISE 2002.