On Making RAMSES an Earth Observation Application Framework[*]

Marcello Mariucci, Bernhard Mitschang
Institute of Parallel and Distributed High-Performance Systems, University of Stuttgart
Breitwiesenstr. 20-22, D-70565 Stuttgart, Germany
firstname.lastname@informatik.uni-stuttgart.de

Keywords: Frameworks for Information Technologies, Software Architectures for Information Systems, Component-Based Designs, Service-Based Approaches.

Abstract

RAMSES is one of the first large-scale prototypes of an operational Earth Observation (EO) application system. It implements a complex infrastructure for the extensive support of a thematic EO application system, which focuses on the detection and monitoring of oil spills. Since EO application systems are usually built on top of a set of generic functions, this paper analyses and assesses the RAMSES infrastructure in order to form a generic EO application framework. This framework should mainly support the collaborative development and customization of emerging EO application systems by maximizing the use of already existing system facilities. Furthermore, it should support the flexible extension and rapid reconfiguration of workflows as the business changes. Results of our analyses show that the RAMSES infrastructure does not cover all requirements of an EO application framework. We therefore introduce advanced design concepts and propose a new framework architecture that structurally controls the inherent complexity of the interdisciplinary domain of EO application systems.

1. INTRODUCTION

Earth Observation (EO) application systems are characterized by the provision of thematic, value-added services, which are based on the intensive use of EO data from space. Single EO data products contain geophysical parameters of our planet and can reach sizes of several hundreds of MBytes. They are acquired and stored in a set of distributed facilities, which manage and disseminate data according to their data policies. The elaboration of these data products requires intensive computation across a series of complex processing steps, and involves several scientific models and auxiliary information. Furthermore, the generation of thematic products requires the collaboration of several competences, which are typically located on geographically distributed sites (see [1]).

The purpose of an EO application framework is to provide a reusable software infrastructure for the production of custom EO application systems. It simplifies the access to generic EO facilities (e.g. EO data archives and image processing modules) and supports the collaborative development of EO application services during the complete software lifecycle. To this end, it breaks down EO application systems into a federation of autonomous and interactive components, relies on their interfaces, and regulates the interactions that they can engage in. Furthermore, it supports the maximal reuse and integration of stabile software elements, workflows and architectures. An EO application framework provides an ideal infrastructure for controlling the inherent complexity of the interdisciplinary EO application domain.

The European Space Agency (ESA) has approached with several activity directions the development of such an EO application framework. One of these activities was to analyze and assess the suitability of one of the first thematic EO application systems to form a generic EO application framework. The idea was to reuse a set of generic functions and to add specific features needed for new thematic EO application services. In Section 2 this thematic EO application system, RAMSES, is introduced, and related activities to analyze and assess the suitability of RAMSES to form a generic EO application framework are described. Furthermore, encountered problems and limitations are pointed out. Based on these experiences, in Section 3 advanced design concepts and a new framework architecture to structurally control the inherent complexity of the interdisciplinary domain of EO application systems are proposed. Section 4 discusses the proposed framework and refers to related work. Section 5 concludes the paper and describes future work.

2. RAMSES

RAMSES (Regional earth observation Application for Mediterranean Sea Emergency Surveillance) is a two-year research project, which developed and validated one of the first prototypes of an operational thematic EO application system. Funded by the European Commission, it demonstrated the effectiveness of distributed, computer-supported EO environmental services in detecting and monitoring oil spills in the Mediterranean Sea. This section briefly summarizes the RAMSES system architecture, and points out its limitations to form an EO application framework.

[*] This work is partially supported by the European Space Agency

2.1 System Architecture

The main service of the RAMSES system is to provide customers, such as the Italian Costal Guard and the Egyptian Authority, with thematic high-level products, which emphasize the existence of oil slicks near their coasts. This service embraces the continuous acquisition and ingestion of data providers' satellite data, the generation of marine and meteorological forecasts, and the corresponding expert analysis. As soon as an oil slick is detected, notifications are sent to subscribed customers, and appropriate product generation routines are activated. In addition, the system offers off-line access to its catalogues, which include a historical storage of all available oil slick related products.

The RAMSES computing infrastructure is based on a modular, object-oriented design. It is divided into several modules, which are deployed on geographically distributed computing machines. The communication between modules is performed by means of a CORBA-conform object request broker (ORB), whereas large datasets are independently transferred by using the file transfer protocol (FTP).

RAMSES actors (e.g. customers or scientific institutions) interface the RAMSES computing infrastructure by means of appropriate 'Remote Access Clients'. These client applications are mainly interactive graphical interfaces, which are directly connected to the communication bus. To access services, they refer to corresponding 'Remote Application Servers' (RAS), which coordinate the workflow of requested services.

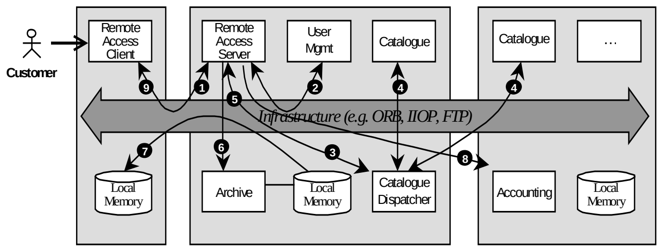

Figure 1: RAMSES production workflow scenario

The following scenario (see Figure 1) illustrates a simplified example of a RAMSES production workflow. A statistical institution, in the role of a RAMSES customer, wants to retrieve all available satellite images of a specific region at a specific time that contain oil slicks. It therefore starts its local 'Remote Access Client' application, specifies the required parameters and requests the service ❶. The RAS, which encodes the workflow model of the requested service, begins by verifying if the user has sufficient rights ❷, it then orders the 'Catalogue Dispatcher' to gather all datasets, which correspond to the user specified parameters ❸. The 'Catalogue Dispatcher' determinates the appropriate system catalogues, forwards the request to them ❹, and sends the datasets back to the RAS ❺. Based on the result, the RAS orders the corresponding archives to retrieve the related images ❻ and to send them to the user-specified location ❼. After all images have been successfully transferred, the RAS orders to the 'Accounting' to charge accrued expenses on customer's bill ❽, and notifies the 'Remote Access Client' that the service has been executed ❾.

For a more detailed description of the RAMSES system see [2-3].

2.2 Limitations to form an EO Application Framework

The RAMSES infrastructure was originally conceived for the support of a single thematic EO application. The main objective was to implement an extensive infrastructure, which supports users during the complete EO value chain. The focus was put on the efficient implementation of system functionalities, rather than on flexibility and openness properties of the system. Thus, extensions and adaptations of the RAMSES infrastructure to support additional EO application services led to several problems and limitations.

First limitations were encountered during the integration of new remote application clients to the infrastructure. Each client application required the implementation of a new server application (i.e. RAS), which defines the workflow model of the services requested by the client. Since the RAS hard-codes these workflow models, implemented workflows could not be easily reused. Furthermore, the approach of hard-wiring workflows in the RAS code showed to be a fragile and not scalable mechanism prone to ambiguities. The larger the workflow and the more actors involved, the more static the process became and the harder it was to adapt the workflow to changes and new conditions. Small modifications or adaptations of the workflow required the recompilation of the RAS code and therefore could only be performed by a small group of developers. Moreover, RAMSES services were not executed under the control of an appropriate workflow management system, which led to data inconsistencies and loss of input information.

Further analyses showed that the modular approach in RAMSES (and in general) was very restrictive to new EO applications. Modules (in the object-orientation paradigm also known as packages) are structured units, which are linked together to form a complete program. Since there is no common way to access modules' content and functionality, their combination and assembling is application-specific and limited to a small group of related developers. In addition, modules support dependency relationships between one another, which makes the substitution of single modules almost impossible and the localization of errors difficult. Workflow descriptions are therefore not strictly separated from their implementations, which in turn makes the RAMSES infrastructure a complex, monolithic and twisted computing system.

Further it has been realized that the transparency of the RAMSES infrastructure is very limited. Many aspects of the EO application system (e.g. architectural design, availability or deployment of software elements and services) are either hard-coded within a wide rage of software packages or described in some documents. This makes it hard for EO application developers to choose and reuse the right software elements and services, and to perform modifications or extensions.

Last but not least, the concurrent and collaborative development during RAMSES infrastructure extensions was awkward and time-consuming. Software updates influenced the whole system by affecting other already stable EO applications. Operational activities of the complete EO application framework were suspended and existing codes overwritten. Furthermore, there was no possibility to simply go back to older versions.

In summary, analyses turned out that the RAMSES infrastructure is extremely inflexible and not particularly suitable to form an EO application framework. The integration of new EO application services introduces a huge amount of overhead, leading to a twisted and hardly manageable software system. Furthermore, required EO application framework facilities, such as support of collaborative development and flexible modification of thematic EO application services, are not provided. As a consequence, an appropriate architectural framework design that structurally controls the inherent complexity of the interdisciplinary domain of EO application systems is required.

3. A SERVICE-BASED COMPONENT FRAMEWORK

Based on the experience gained in the extension and adaptation processes of the RAMSES infrastructure, we propose a new design approach for an EO application framework. To distinguish it from other EO application framework design approaches, we call it ARSENAL framework. ARSENAL (Study on a Repository Supported Earth Observation Application Framework) is an ESA funded project at the University of Stuttgart (see [4]).

Our vision includes a shared information database on the structure and flows of EO applications. This knowledge database mainly contains descriptions of the structured collection of components, which are collaboratively assembled to provide EO application services. Therefore, it also contains control and data flow specifications, which can be flexibly extended and rapidly reconfigured as the business changes. Furthermore, it includes system architecture models, description of system behaviors, constraints, access structures and deployment information required for the suitable management of today's software infrastructures.

In the following sections, basic elements of our approach are introduced. First, EO application components which represent the primitive elements of the ARSENAL framework are introduced. They are then assembled to EO application services, which are (among others) managed by an EO application repository. These concepts are finally combined to an overall ARSENAL framework architecture, which is described and discussed.

3.1 EO Application Components

EO application components (EO AppComps) of the ARSENAL framework are isolated software blocks that

- offer a specific, common accessible functionality,
- represent reusable, logically self-contained application concepts, and
- are independently developed, tested, maintained and deployed.

EO AppComps are commonly accessed through interface structures, and easily plugged and substituted within the ARSENAL framework. Examples of EO AppComps are satellite data archiving systems, invoicing systems, image processing units, and GUIs for thematic analyses.

Typically, EO AppComps are coarse-grained components that are built in total isolation by different companies. They should not be interpreted as dynamic run-time instantiations such as objects, but rather understood as complex systems (i.e. applications) whose functionality is covered by stabile interfaces. EO AppComps are either implemented by human activity, or by a federation of implementation objects, procedures and low-level components (e.g. CORBA components, Enterprise JavaBeans), or a combination of both.

EO AppComps are therefore prefabricated, interactive units that offer certain functionalities. They do not act alone but are employed in EO application services (see section 3.2) typically by arranging their functions in composites. Building services by combining stabile EO AppComps improves quality and supports rapid development, leading to a shorter market time. At the same time, adaptations to changing requirements can be achieved by investing only in key changes of a component, rather than undertaking a major service release change. Therefore, the EO AppComp approach seems to be ideal for current service-based system architectures.

EO AppComps strictly separate their interface specification from their implementation. These EO AppComp specifications structurally and semantically describe EO AppComps and their functions, and link to the corresponding implementations. Thus, EO AppComp specifications can be managed and accessed independently of their realization, which can be deployed on several dispersed machines. In addition, EO AppComp specifications are neutral to any technology and vendor specific implementations. Consequently, EO AppComp implementations can be substituted and maintained without affecting the related specification. Furthermore, legacy systems can be easily wrapped into EO AppComp specifications and hence made available for EO application service composites.

3.2 EO Application Services

EO application services (EO AppServs) of the ARSENAL framework are logically self-contained EO application-specific production tasks. They are composed of three elements, namely

- an *interface*, which specifies the service access structure,
- a *set of functions*, which implements the service, and

- a flexible specification of *control and data flows*, which defines the collaboration between functions in order to execute the service.

As EO AppComps, EO AppServs represent reusable, logically self-contained application concepts, which are commonly accessed through service structures. The difference between EO AppComps and EO AppServs is that EO AppComps constitute an implementation of a set of functions, whereas EO AppServs typically specify relationships between functions in order to provide a single service to customers. EO AppComps can be seen as *black boxes*, which cover their internal representations and implementations. EO AppServs, however, are more like *grey boxes*, which indicate their internal composition and relationships. Examples of EO AppServs are user subscriptions, thematic analyses, product requests, and activity plan deliveries.

EO AppServs consist of a stabile interface and a description of an internal data flow and control flow. These '*service flow*' descriptions define relationships between function calls, which are implemented by EO AppComps. EO AppServs are neutral to any technology and vendor-specific implementation, which enables different EO AppComps work together.

Primitive elements of EO AppServ compositions are EO AppComps functions. EO AppServ compositions do not consider user interactions, since human activities are always covered by EO AppComps. EO AppServ compositions are linked structures, which include recursive, associative and transitive relationships between EO AppServs. The EO AppServ processing environment is responsible for their proper execution.

3.3 EO Application Repository

Repository technology (see [5]) provides an integrated storage area and appropriate management and access services for domain-specific metadata. It can be seen as an advanced database management system responsible for the storage, sharing and management of descriptive information, i.e. metadata. Indeed, in addition to basic database management functions, the repository manager provides indispensable functions for the collaborative management of system metadata (e.g. workspaces, contexts). Furthermore, it offers functions for the seamless management of metadata throughout the complete software lifecycle (e.g. versioning, configurations). The main value of the repository technology and its functions are summarized in [6].

The ARSENAL framework EO application repository (EO AppRep) provides repository functions in order to manage and integrate EO application metadata. This metadata mainly consists of EO AppServs and EO AppComp descriptions, as well as administrative information (e.g. user profiles, accounting), data structures, dependencies, and general descriptions of the EO application architectural design. The intention is to form a knowledge information base about the EO application framework's artifacts in order to foster agile and collaborative work across the EO application development and operational lifecycle. Furthermore, it supports metadata consistency, integrity and reuse. Hence, it reduces the time needed to find the appropriate information, decreases the time to market, and, most importantly, effectively shares the information across people, tools and software components.

3.4 Repository-supported, Service-based Component Framework for EO Applications

The proposed ARSENAL framework provides an advanced infrastructure for the seamless development, integration and execution of responsive EO application system services. It relies on a service-based component architecture, and manages system metadata by means of a repository system. It includes an access portal for *client access*, an EO AppRep for *service development* and *management*, a workflow management system for *service execution*, and a set of (interactive) EO AppComps for *service implementation*. Furthermore, it integrates middleware services for transparent, reliable and secure *communication*. Figure 2 depicts the ARSENAL framework architecture. The highlighted EO AppServ describes the production workflow scenario of Figure 1. Each building block represents an EO AppComp activity, which links to its related function implementation through the middleware.

Whereas in RAMSES the client sends the service request to a RAS, in the ARSENAL framework the client accesses the portal to obtain the service ❶. The request is then forwarded to the EO AppRep ❷, which identifies the related EO AppServ flow model, and verifies that the customer is authorized to access included EO AppComp functions. Afterwards, it sends corresponding workflow instructions to the workflow management system ❸. The control flow of the EO AppServ calls up the 'Catalogue Dispatcher' and the other EO AppComps ❹. It then gives the control back to the workflow management system ❺. The workflow management system sends the results back to the client by means of the access portal ❻.

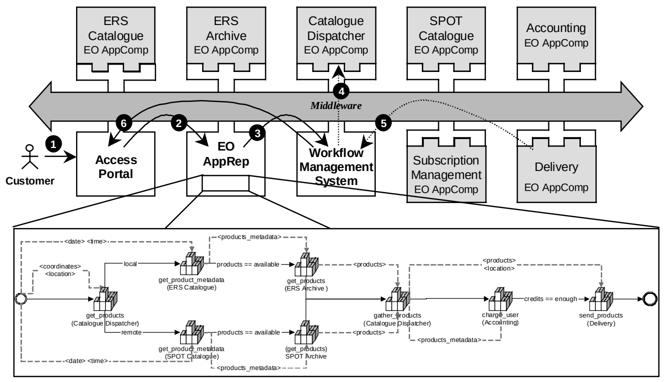

Figure 2: ARSENAL framework production workflow scenario

The EO AppRep does not include the physical implementation of EO AppComps. It manages their metadata and supports retrieval and access to EO AppComp implementations.

Even though this concept could have a negative impact on the consistency and integrity of the framework (e.g. the EO AppRep still links to a removed EO AppComp), it is not feasible to integrate huge EO AppComps like archive systems into the EO AppRep. Furthermore, hardware dependencies and security precautions of software providers often do not allow the modification and flexible deployment of codes. For this reason EO AppComp activities are linked to their implementation, and functionalities are remotely accessed through their interfaces.

EO AppServs are completely managed by the EO AppRep. Several EO AppServs configurations can thus be simultaneously treated and easily adapted to changes or extensions. Furthermore, EO AppServ flows comply with workflow descriptions. They can thus be executed by means of a general-purpose workflow management system, which supports their automation and consistent processing.

4. DISCUSSION AND RELATED WORK

The basic idea of the proposed ARSENAL framework is to integrate coarse-grained components (i.e. applications), and to seamlessly support the so-called *programming in the large* (see [7]) in EO application systems. For this purpose it relies on a model driven architecture (see [8]), and controls relationships between components by means of a workflow management system.

We claim that the ARSENAL framework represents an advanced platform for supporting multiple, interdisciplinary EO application systems during the complete software lifecycle. Unlike the RAMSES infrastructure, the ARSENAL framework provides an integrated mechanism to manage the complexity of EO application systems. Specifications of available system components and their relationships are consistently stored and managed by means of an application repository system. System components can therefore be easily reused and put in different contexts, i.e. EO application services. Furthermore, EO application services can be collaboratively created, as well as flexibly modified, extended and customized. Whereas in the RAMSES approach the extension or modification of application services implies a recompilation of software codes, the ARSENAL framework separates service and component specification from their implementations. In this way, application services can be dynamically reconfigured and reliably executed by means of a workflow management system.

The drawback of the ARSENAL framework compared to the RAMSES approach is the loss of performance in executing EO application services. Whereas in RAMSES workflows are hard-coded and precompiled, in the ARSENAL framework first EO application services have to be retrieved in the EO AppRep and then sent to the workflow management system for execution. However, the execution of EO application services is generally not time critical. EO application services typically integrate complex, time-consuming processing steps and involve several EO application component activities including user interactions. Consequently, this drawback does not considerably influence the overall system and therefore can be left out of consideration. Table 1 summarizes the comparison between the ARSENAL framework and the RAMSES approach.

Table 1: RAMSES approach vs. ARSENAL framework

	RAMSES approach	ARSENAL framework
Manageability	--	++
Reuse	+	++
Integration	+	++
Extensibility	-	++
Flexibility	--	++
Collaborative Development	--	+
Reliability (execution control)	--	++
Fault tolerance (recovery)	-	+
Execution efficiency	+	-

Rather than a new technology, the proposed ARSENAL framework represents an architecture, which combines technological concepts for the flexible management of the complex structure of EO applications. The analysis of related implementation aspects and development methodologies is beyond the scope of this paper.

Component-based application frameworks have attracted attention from many researchers and software engineers. Thus, component-based application frameworks have been evaluated, defined and successfully employed in many domains (see [9-11]). However, the proposed ARSENAL framework mainly distinguishes from them by the definition and granularity of its components and services, as well as by the integration of a repository and a control-based workflow management system. System metadata is extracted from code and uniformly managed by advanced database technology. In addition, system services are collaboratively developed, and their control and data flows managed by workflow technology.

Components, as defined in the ARSENAL framework, are similar to 'system-level components' as described in [12]. The conceptual difference between these two approaches is their transparency. Whereas EO AppComps are seen as black boxes, system-level components are also concerned with their internal composition.

Finally, the ARSENAL framework can be compared to current application integration platforms as described in [13-15]. However, these process driven platforms typically include "build-in" repositories based on files or LDAP trees. Interfaces to these proprietary repository implementations are optimized to their applications, and are not designed to interoperate with external repositories. Furthermore, the metadata model is specific to some tools (e.g. workflow system) and do not represent a uniform view of the system.

5. CONCLUSION AND FUTURE WORK

In this paper we showed that the RAMSES infrastructure is extremely inflexible and not particularly suitable to form an EO application framework. Based on the experience gained in this analysis, we proposed a new architectural design of an EO application "integration" framework (we called it ARSENAL framework) for managing the complexity of the interdisciplinary domain of EO applications. Our main contribution is the service-based component architecture that builds on component, service, repository and workflow concepts. We see our framework architecture as offering significant advantages over the RAMSES solution, as it provides a better flexibility and collaboration support in developing and customizing EO application systems. Main benefits are described in more detail in the following list.

- *Management of complexity*: The ARSENAL framework divides large-scale applications into well-defined, logically self-contained software components, and enables the rapid and efficient assembly and deployment of new applications. It is therefore a good mechanism to manage complex systems such as EO applications ones. Furthermore, it defines standards for EO software component interfaces.
- *Flexibility/Agility*: The ARSENAL framework strictly separates service and component specifications from their implementations. Therefore, EO application systems can be independently engineered by using any implementation technology, and services can be easily reconfigured and customized as the business changes.
- *Reuse and Integration*: EO AppServs describe their internal composition by linking to EO AppComp implementations through their interfaces. Therefore, EO AppComp implementations can easily be substituted and shared. Furthermore, any legacy system can be easily wrapped and integrated in the EO application framework.
- *Metadata resource management*: ARSENAL framework metadata, such as EO AppComp descriptions and EO AppServs, are managed by an application repository system. This ensures metadata consistency and integrity, and allows the establishment and query of relationships between structures.
- *Collaboration*: The EO AppRep supports the management of concurrent activities in the EO AppServ design and development. Thus, EO application developers can asynchronously work together by exchanging results across the entire development lifecycle.
- *Execution control*: EO AppServs are executed, monitored and controlled by means of a workflow management system. EO AppServs are therefore reliably executed by enforcing and monitoring the processing of each specified step.

Future work is related to several issues of the ARSENAL framework. We are defining the required functions and information model of the EO AppRep that properly fits to the EO application framework. Furthermore, we will analyze implementation aspects of the ARSENAL framework by using the “Service Web” [16] and Grid [17] technologies.

ACKNOWLEDGMENT

The ideas described in this paper are the result of the research effort carried out within the ARSENAL project (see [6]). We would like to thank Luigi Fusco, Günther Landgraf and Christoph Mangold for their financial and intellectual support.

REFERENCES

[1] Möller, H.L.; Mariucci, M.; B. Mitschang. 1999. “Architecture Considerations for Advance Earth Observation Application Systems.” In *Proceedings of the Second Interoperating Geographic Information Systems Conference* (Zurich, CH, March 10-12). INTEROP’99, Springer, 75-90.

[2] Mariucci M.; Caspar, C.; Fusco, L.; Henaff, Y.; M. Forte. 2000. “RAMSES: An Operational Thematic EO-Application on Oil Spill Monitoring. System description.” In *Proceedings of the Earth Observation & Geo-Spatial Web and Internet Workshop 2000 Conference* (London, UK, April 17-19). EOGEO2000, http://webtech.ceos.org/eogeo2000.

[3] Matra Systems & Information. 1998. “Regional earth observation Application for Mediterranean Sea Emergency Surveillance: RAMSES.” *EC ESPRIT Program*, Project proposal, ESPRIT-1998-28245, http://ramses.esrin.esa.it.

[4] European Space Agency. 2001. “study on A Repository Supported Earth observatioN AppLication framework: ARSENAL.” *ESA Program*, Ref. GRID-GSR-001-SOW, http://www.informatik.uni-stuttgart.de/ipvr/as.

[5] Bernstein, P.A.; U. Dayal. 1994. “An Overview of Repository Technology.” In *Proceedings of the 1994 On Very Large Databases Conference* (Santiago, Chile, Sep.12-15). VLDB, Morgan Kaufmann, 705-713.

[6] Bernstein, P.A. 1997. “Repositories and Object Oriented Databases.” In *Proceedings of the Datenbanksysteme in Büro, Technik und Wissenschaft Conference*. BTW, Springer, 34-46.

[7] Brown, A.W. 2000. *Large-scale, Component-based Development*. Prentice-Hall.

[8] Soley, R. et al. *Model Driven Architecture*. White Paper. Object Management Group. http://www.omg.org.

[9] Fingar, P. 2000. “Component-based Frameworks for E-Commerce.” *Communications of the ACM 43*, No 10, October:61-67.

[10] Larsen, G. 2000. “Component-based Enterprise Frameworks.” *Communications of the ACM 43*, No 10, October:24-26.

[11] Fayad, M.E.; R.E. Johnson. 2000. *Domain-Specific Application Frameworks*. John Wiley & Sons.

[12] Herzum, P.; O. Sims. 2000. *Business Component Factory. A Comprehensive Overview of Component-based Development for the Enterprise*. John Wiley & Sons.

[13] Microsoft Corporation. *BizTalk Server 2000 Product Overview*. http://www.microsoft.com/biztalk.

[14] IBM Corporation. *Websphere Software Platform Product Overview*. http://www.ibm.com/websphere.

[15] SAP Corporation. *Integrated E-Business Platform Product Overview*. http://www.sap.com

[16] Ryman, A. 2000. *Understanding Web Services*. IBM report. http://www7.software.ibm.com.

[17] Foster, I.; C. Kesselman. 1999. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann.