Building Distributed Access Control System Using Service-Oriented Programming Model

Ivan Zuzak, Sinisa Srbljic
School of Electrical Engineering and Computing, University of Zagreb, Croatia
ivan.zuzak@fer.hr, sinisa.srbljic@fer.hr

Ivan Benc
Ericsson Nikola Tesla d.d., Zagreb, Croatia
ivan.benc@ericsson.com

Abstract

Service-Oriented Programming Model is a new methodology for building service-oriented applications. In the Service-Oriented Programming Model, an application is assembled from loosely coupled, dynamically bounded services that mutually cooperate and compete in order to achieve the application's goal.

In this paper, we present our experiences with the Service-Oriented Programming Model attained through development and testing of Access Control System. The Access Control System is a service-oriented application that manages security in the virtual organizations. We indicate how Service-Oriented Programming Model facilitates run-time reconfiguration of the Access Control System. Further, we evaluate and compare the performance of several architectures of the Access Control System with different levels of parallelism and concurrency.

1. Introduction

Service-oriented computing [1] has recently emerged as a new paradigm for creating distributed applications. The service-oriented computing creates applications using the service-oriented architecture [2], an architectural style that builds applications from loosely coupled, dynamically bounded services. The service-oriented architecture introduces a new level of flexibility because it uses services as implementation independent software components that can be dynamically discovered, composed and orchestrated into new services or applications.

Service-Oriented Programming Model (SOPM) [8] is a new methodology for the design, development, and execution of service-oriented applications. The SOPM utilizes *Coopetition Based Distributed Architecture.* In the Coopetition Based Distributed Architecture, application services mutually cooperate and compete without central control authority in order to reach the global goals of the application. The *Programmable Internet Environment*[1] (PIE) [9] is an environment that implements the SOPM and enables development and execution of SOPM-based applications over Internet infrastructure.

In this paper, we present the application of the SOPM in a service-oriented *Access Control System* [7]. Our goal while building the Access Control System is to explore the potentials of the run-time reconfiguration of the SOPM-based applications. Particularly, we investigate how various application architectures with different levels of parallelism and concurrency influence the overall performance of the SOPM-based application.

The remainder of the paper is organized as follows. Service-Oriented Programming Model is described in section II. Section III presents the Access Control System built using SOPM. Section IV gives a performance analysis of the Access Control System, while section V gives a conclusion.

2. Service Oriented Programming Model

The Service-Oriented Programming Model (SOPM) is a new methodology for building service-oriented applications. The SOPM is based on the following principles: *Coopetition Based Distributed Architecture*, *End-user development framework* and *Distributed translation process*.

In the *Coopetition Based Distributed Architecture* (CBDA) applications are composed of services that perform local actions without central control authority. The term coopetition designates that the services simultaneously cooperate and compete while executing application goals. Figure 1 presents an application based on the CBDA. The application consists of *Application*

[1] The Service-Oriented Programming Model, Coopetition Based Distributed Architecture, Access Control System, and Programmable Internet Environment are developed at the School of Electrical Engineering and Computing at the University of Zagreb, Croatia, in cooperation with Ericsson Nikola Tesla d.d., Zagreb, Croatia, and cosponsored by Ministry of Science, Education, and Sports through the national Cro-Grid project. More information is available at: http://www.ris.fer.hr, http://www.pie.fer.hr and http://www.cro-grid.hr.

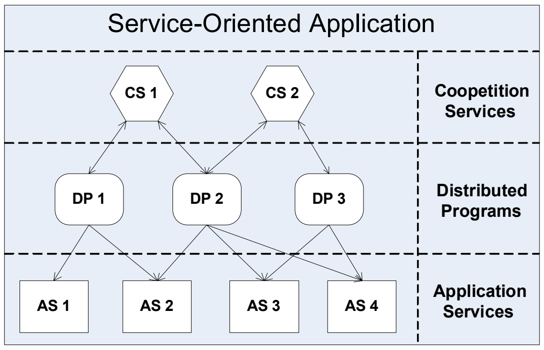

Figure 1. Coopetition Based Distributed Architecture

services, *Coopetition services* and *Distributed programs.*

The *Application services* implement coarse fragments of application's computational logic. These services are delivered on-demand through globally accessible network. Alternatively, Application services can be custom developed in order to provide specific functionalities. *Coopetition services* [3] are pre-built services of the SOPM environment that are used for coordination and synchronization of the Application services. These services are Semaphore, MailBox, and EventChannel. For instance, MailBox supports the persistent asynchronous communication between services using message-oriented communication model. *Distributed programs* [4] handle cooperation and competition between Application services. They use Coopetition services in order to bind and synchronize Application services into a distributed application. Distributed programs are specified in a process description language CL (Coopetition Language) [5], which consists of the subset of BPEL4WS [11] and WSDL languages [12].

The *End-User development framework* is a paradigm that simplifies development process of distributed applications founded on Coopetition Based Distributed Architecture. The End-User development framework is based on the *Simple Services Composition Language* (SSCL) [13], a service composition language targeted at end-users. The end-users use SSCL to build Distributed programs. The *Distributed translation process* transforms Distributed programs written in SSCL into programs written in CL. Further, is sets-up the CL programs for execution on the computers in the SOPM environment.

The *Programmable Internet Environment* (PIE) [9] is environment that implements SOPM. PIE supports development and execution of SOPM-based applications over Internet infrastructure. Specifically, PIE implements the coopetition services, the interpreter for the CL distributed programs, and the translation process for distributed programs written in the SSCL. In addition, PIE offers management and deployment tools [5] for application and coopetition services and the management

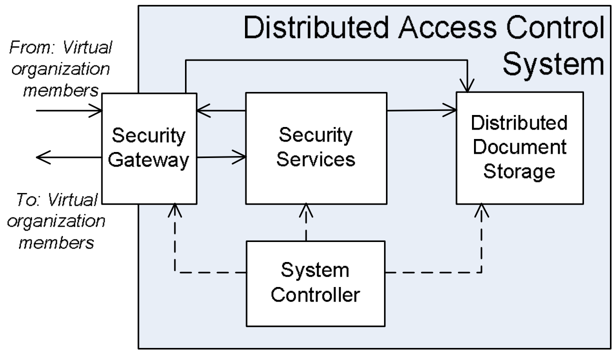
Distributed Access Control System
From: Virtual organization members
To: Virtual organization members
Security Gateway
Security Services
Distributed Document Storage
System Controller

Figure 2. The overall organization of the distributed Access Control System

tool for the SSCL translation process. Furthermore, PIE creates a virtual network of service providers. Applications written for the PIE utilize application services from the virtual network and services publicly available on the Internet.

3. Access Control System

The Access Control System is a part of the infrastructure supporting virtual organizations. A *Virtual organization* [10] is a group of users and services that form a community in which services are used in a secure, reliable and accountable manner. The Access Control System operates as organization's controlling authority that authenticates members of the virtual organization, and secures and authorizes access to the services in the organization.

Figure 2 presents the overall organization of the distributed Access Control System. The system consists of four subsystems: *Security Gateway*, *Security Services*, *Distributed Document Storage*, and *System Controller*.

Distributed Document Storage (DDS) is a scalable, fault-tolerant, service-oriented storage system designed for storage of XML documents. Other subsystems of the distributed Access Control System use the DDS to store and fetch XML documents required in the operation of the virtual organization. The *Security Services* subsystem executes registration and authentication services of the virtual organization. These services enable users and other services to join and authenticate with the organization. The *Security Gateway* acts as a policy enforcement point that controls the access to the services exposed in the virtual organization. All service calls in the virtual organization are tunneled through the Security Gateway, which authorizes access to the services. The *System Controller* is an administrative subsystem that deploys, configures, and reconfigures distributed Access Control System over a set of available computers.

3.1. Distributed Document Storage

Figure 3 presents the structure of *the Distributed Document Storage* (DDS). The Distributed Document

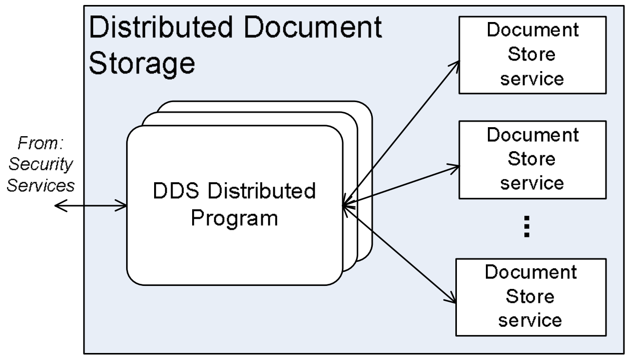

Figure 3. The organization of the Distributed Document Storage subsystem

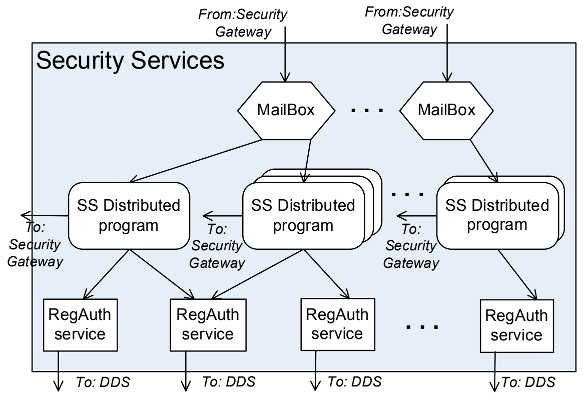

Figure 4. The organization of the Security Services subsystem

Storage consists of *Document Store* services and *DDS Distributed programs*. *Document Store* services store and fetch XML documents from their local storage. DDS Distributed programs coordinate the work of the Document Store services.

DDS Distributed programs implement the coordination logic of the DDS. The coordination logic is based on a mapping table encoded into the bodies of distributed programs. Each time a DDS Distributed program receives a document request, it uses the encoded mapping table to map the request to the Document Store service that stores a copy of the requested document. Distributed program forwards the request to the selected Document Store service, receives the response, and forwards the response to the requestor. The DDS maintains the consistency of the copies of documents by synchronizing document modifications using the two-phase commit protocol.

The structure of the Distributed Document Storage is reconfigurable. Access Control System administrators can modify the number of Document Store services, mapping of the documents to the Document Store services and the number of DDS Distributed programs.

3.2. Security Services

Figure 4 presents the structure of the *Security Services* subsystem. The subsystem consists of a set of *MailBoxes, RegAuth* services and *SS Distributed programs. MailBox* services are Coopetition services of the SOPM environment. Each MailBox serves as a queue of requests that are pending for processing. The *RegAuth* services are SOPM Application services that implement the actual request processing logic. Each RegAuth service implements the logic for processing both the registration and authentication requests. The *SS Distributed programs* are SOPM Distributed programs that obtain requests from MailBoxes and forward them to RegAuth services. Each SS Distributed program is defined by three parameters: the Mailbox it uses to obtain the requests, the set of RegAuth services it forwards the requests to, and the number of instances it runs concurrently. Number of instances of the SS Distributed program defines the number of requests it obtains and processes concurrently.

Members of the virtual organization put registration and authentication requests into MailBoxes through the Security Gateway subsystem. The Security Gateway forwards each request to one of the MailBoxes. One instance of a SS Distributed program obtains the pending request from the MailBox and forwards it to one of the RegAuth services. The RegAuth service decrypts, verifies, and processes the request. During processing, the RegAuth service fetches, stores, deletes and updates documents in the DDS. The SS Distributed program receives the response from the RegAuth service and forwards it to the user through the Security Gateway.

The structure of the Security Services subsystem is configurable. Administrators can modify the number and placement of MailBoxes, RegAuth services, and SS Distributed programs. Furthermore, they can modify the parameters of the SS Distributed programs.

3.3. Security Gateway

The *Security Gateway* subsystem consists of several *Access* services. Each *Access* service operates as an independent security-gateway that receives and forwards service requests according to the security policies and credentials of the virtual organization. The Access services fetch the security policies and credentials from the DDS. Since Access services execute mutually independently, there is no coordination logic in the Security Gateway subsystem. Administrators can modify the number of Access services in order to increase or decrease the throughput of the Security Gateway subsystem.

3.4. System Controller

The System Controller is an administrative subsystem that configures and reconfigures the distributed Access

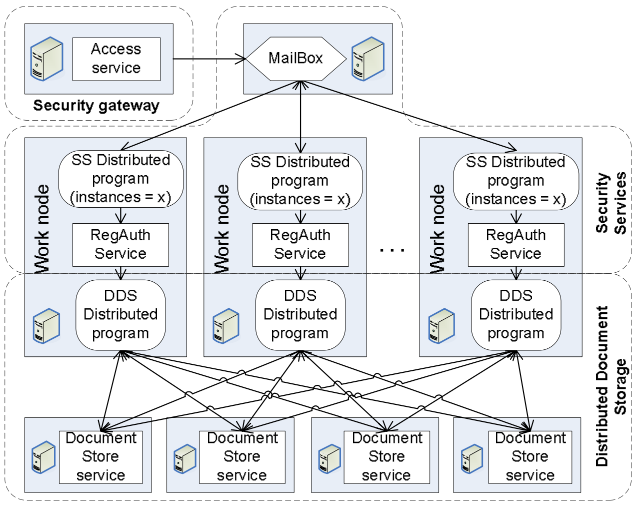

Figure 5. Pull architecture of the distributed Access Control System

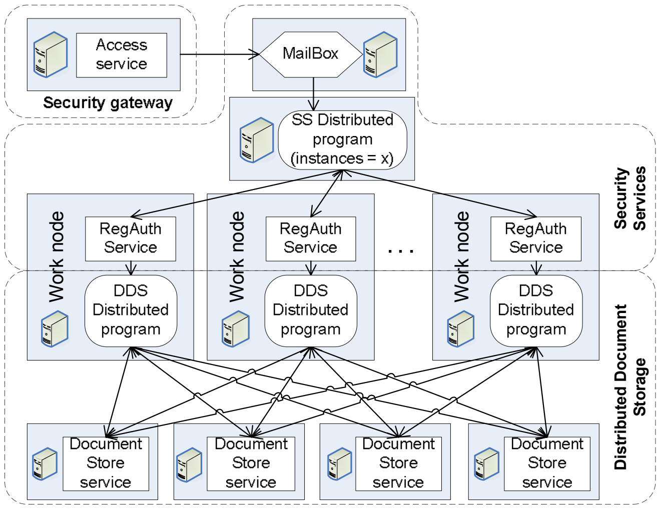

Figure 6. Push architecture of the distributed Access Control System

Control System. The configuration and reconfiguration processes are based on the *Access Control System Model*. This model consists of a set of rules that define the allowable configurations of the distributed Access Control System. For instance, the model specifies that each SS Distributed program is connected to exactly one MailBox, while it can forward requests to several RegAuth services.

Administrators start the reconfiguration process by sending new configuration document to the System Controller. During reconfiguration, the System Controller uses predefined Distributed program templates and supplied target configuration to generate SS and DDS Distributed programs. Further, the System Controller uses PIE installation mechanisms to deploy application services, coopetition services and generated Distributed programs over a set of computers available in the distributed environment.

4. Performance Analysis

The goal of the analysis is to determine the effect different system architectures have on the performance of the Access Control System. The analysis is based on the measurements done with stress testing application. The testing application sends authentication requests to the Access Control System and measures processing time. The testing application maintains a constant load on the Access Control System. In particular, it maintains a fixed number of requests processed in the system parallelly.

In this article, we present measurement results for two architectures of the distributed Access Control System: the *Pull architecture* and the *Push architecture*. Figure 5 presents the *Pull architecture.* In this architecture, the Security Gateway subsystem runs on one machine, while the Security Services and Distributed Data Storage subsystems are distributed. The machines denoted as *Work nodes* execute a RegAuth service, a SS Distributed program, and a DDS Distributed program. Each SS Distributed program uses local RegAuth service to process the requests, while the RegAuth service uses local DDS Distributed program to access the Document Store services. We named this architecture Pull architecture because work nodes pull the requests from the mailbox. Thus, they are competing for the processing of the requests. Different configurations of the Pull architecture are formed by modifying the number of active work nodes and the number of instances of SS Distributed program on each work node.

Figure 6 presents the *Push architecture*. In this architecture there is only one SS Distributed Program pulling the requests from the mailbox. The SS Distributed program pushes the requests to several work nodes using the round robin algorithm. The work nodes in this architecture are lighter than in the Pull architecture because they do not contain the SS Distributed program. Different configurations of Push architecture are formed by modifying the number of active work nodes and number of instances of the SS Distributed program.

In the analysis, we also compare against the processing times of the monolithic architecture of the Access Control System [6]. The monolithic architecture executes on two machines: the Web server that processes authentication requests and the back-end database server that stores virtual organization's records.

4.1. Pull architecture test

Figure 7 and Figure 8 present the results for the Pull architecture. Figure 7 presents the processing time as a function of the increasing parallelism in the system, while Figure 8 presents the processing time as a function of the

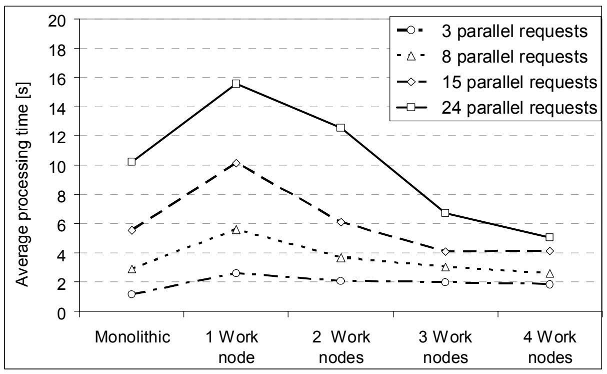

Figure 7. Processing times of Pull architecture configurations with different levels of parallelism

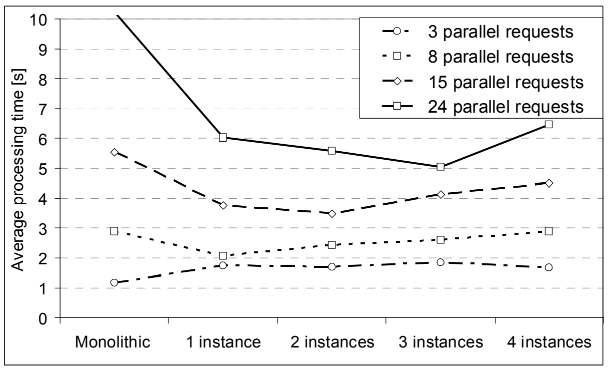

Figure 8. Processing times of Pull architecture configurations with different levels of concurrency

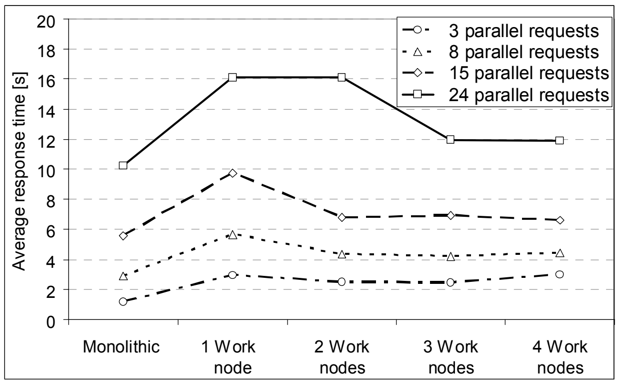

Figure 9. Processing times of Push architecture configurations with different levels of parallelism

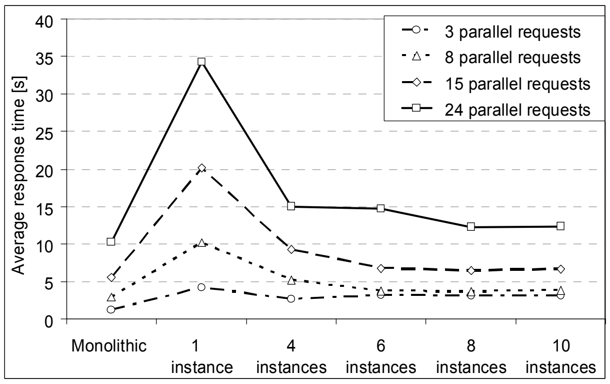

Figure 10. Processing times of Push architecture configurations with different levels of concurrency

increasing concurrency. Different levels of parallelism are achieved by varying the number of active work nodes, with the number of instances of SS Distributed programs fixed at three. Different levels of concurrency are achieved by modifying the number of instances of SS Distributed programs, with the number of work nodes fixed at four. Additionally, both graphs show the results for the monolithic Access Control System.

Figure 7 shows that when only three parallel requests are processed in the system, there is no significant difference in the average processing time of various configurations of the Pull architecture and the monolithic version of the system. This happens because when there is a light load on the system, there are not enough requests to fully utilize all of the available work nodes. Furthermore, while there is a light load on the system the monolithic architecture achieves shortest processing time since it does not have the large service communication overhead of the distributed version. However, as the number of parallel requests sent to the system increases, the configurations with more work nodes achieve shorter processing times. This happens because these configurations can process more requests in parallel.

Figure 8 shows that modifying concurrency does not influence the processing time significantly. However, there is an optimal number of instances per work node and this number depends on the system workload. For light workloads the optimal number of instances is smaller then the one for the heavy workloads.

4.2. Push architecture test

Figure 9 and Figure 10 present the results for the Push architecture. Figure 9 presents the processing time as a function of the increasing parallelism in the system, while Figure 10 presents processing time as a function of increasing concurrency. Different levels of parallelism are achieved by varying the number of work nodes of the Push architecture, with the number of instances fixed at ten. Different levels of concurrency are achieved by varying the number of instances of SS Distributed program, with the number of work nodes fixed at four.

Figure 9 demonstrates that increasing parallelism in the Push architecture does not influence the processing time significantly. This occurs because SS Distributed program does not have the knowledge regarding the workload on the work nodes and forwards requests using the round robin algorithm. Thus, the overloaded and the underutilized nodes get the same number of request to process. Furthermore, since there are a fixed number of requests that SS Distributed program handles concurrently, an underutilized node must wait for the an overloaded node to finish processing of a request before the SS Distributed program can assign it a new request. This scenario makes Push architecture unstable as it leads to the unbalanced load of the work nodes.

Figure 10 shows that the processing time decreases as the number of SS Distributed program instances increases. This occurs because the number of instances dictates the number of requests processed concurrently. For instance, if there is only one instance of SS Distributed program then only one request is processed concurrently. Consequentially, if there are a small number of instances and large number of work nodes, some work nodes are underutilized. However, if there are too many instances, there is no further improvement in the processing time due to the unbalanced work nodes problem described previously.

4.3. Comparison of the architectures

While comparing the processing times of the Push, Pull and monolithic architectures, it is notable that monolithic architecture achieves better processing times when the number of parallel requests is small. However, under heavier workloads distributed Push and Pull architectures perform better. However, the processing time of the Pull architecture is nearly half of the processing time of the Push architecture. This happens because Pull architecture follows principles of Coopetition Based Distributed Architecture more closely then the Push architecture. Specifically, in the Pull architecture each work node competes for the processing of the requests by pulling request from the mailbox. Simultaneously, all work nodes are cooperating in fulfilling the overall goal of the system. On the other hand, in the Push architecture, a central SS Distributed program assigns tasks to the work nodes, thus there is no competition.

5. Conclusion

In this paper, we demonstrate the application of the Service-Oriented Programming Model (SOPM) in the design of the distributed Access Control System. We present two SOPM-based architectures of the distributed Access Control System called Push and Pull architecture. Furthermore, we show how templates of SOPM distributed programs can be utilized for run-time reconfiguration of the presented architectures.

Through experiments, we demonstrate that under heavy workloads Push and Pull architectures achieve better performance than the monolithic architecture of the Access Control System. Moreover, we show that the Pull architecture achieves nearly 50% better processing time than the Push architecture. This occurs because in the Push architecture, central entity assigns request processing to services, while in the Pull architecture, services of the system compete for the processing of the requests.

6. References

[1] M. P. Singh and M. N. Huhns: “Service-Oriented Computing: Semantics, Processes, Agents”, John Wiley & Sons Ltd, 2005.

[2] M. P. Papazoglou and D. Georgakopoulos: “Service Oriented Computing”, Communications of the ACM, Volume 46, Issue 10, October 2003, pp. 25-28.

[3] A. Milanovic, S. Srbljic, D. Skrobo, D. Capalija, and S. Reskovic: “Coopetition Mechanisms for Service-Oriented Distributed Systems“, Proceedings of the 3rd International Conference on Computing, Communication and Control Technologies, Vol. I, Computer Technologies, Austin, Texas, USA, 2005, pp. 118-123.

[4] D. Skrobo, A. Milanovic, and S. Srbljic: “Distributed program Interpretation in Service-Oriented Architectures“, Proceedings of the WMSCI 2005, The 9th World Multi-Conference on Systemics, Cybernetics and Informatics, Vol. IV, Computer Techniques, Orlando, Florida, USA, 2005, pp. 193-197.

[5] M. Podravec, I. Skuliber, and S. Srbljic: “Service Discovery and Deployment in Service-Oriented Computing Environment“, Proceedings of the WMSCI 2005, The 9th World Multi-Conference on Systemics, Cybernetics and Informatics, Vol. III, Architectures, Tools and Distributed Systems, Orlando, Florida, USA, 2005, pp. 5-10.

[6] I. Benc, M. Stefanec, and S. Srbljic: “Usage-tracking by Public Information System Mediator”, in Proceedings of the IEEE MELECON 2004, Croatia, 2004, pp. 723-726.

[7] S. Srbljic et al: “Service Development and Application Integration with Public Information System Mediator”, in Proceedings of the IEEE MELECON 2004, Croatia, May 2004, pp. 713-718.

[8] A. Milanovic: “Service-Oriented Programming Model”, Ph.D. thesis, School of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia, December 2005, original title in Croatian “Programski model zasnovan na uslugama”.

[9] S. Srbljic et al.: “Programmable Internet Environment – PIE”, School of Electrical Engineering and Computing, Zagreb and Ericsson Nikola Tesla, Zagreb, www.pie.fer.hr

[10] I. Foster, C. Kesselman, and S. Tuecke: ”The Anatomy of the Grid Enabling Scalable Virtual Organizations”. Lecture Notes in Computer Science, vol. 2150, pp. 1-4, 2001.

[11] T. Andrews et al.: “Business Process Execution Language for Web Services (BPEL4WS)”, Microsoft, IBM, Siebel Systems, BEA, and SAP, version 1.1, May 2003, http://www.ibm.com/developerwors/webservices/library/ws-bpel.

[12] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana: “Web Services Description Language (WSDL)”, Microsoft and IBM version 1.1, March 2001, http://www.w3.org/TR/wsdl.

[13] I. Gavran, A. Milanović, and S. Srbljić: "End-User Programming Language for Service-Oriented Integration", 7th Workshop on Distributed Data and Structures, Santa Clara, CA, USA, January 2006