



i-Coin

General non-technical platform definition



Sami Laaksonen

General Description
i-Coin

20.6.2007



HISTORY OF THIS DOCUMENT

Version and date	Author	Description
1.0 / 22.3.2007	Sami Laaksonen	First version was created.
1.1 / 27.3.2007	Teemu Karvonen	Document was updated.
1.2 / 28.3.2007	Juhani Talvela	Updated some non-technical details

VERSION OF THIS DOCUMENT

Document	Version and date	Status	File
General platform definition (SOA)	1.0 / 22.3.2007	Ready	General platform definition.doc
General platform definition (SOA)	1.1 / 27.3.2007	Ready	General platform definition.doc



Sami Laaksonen

General Description
i-Coin

20.6.2007



ICT Solution Provider

PROPENTUS



CONTENTS

1	INTRODUCTION	5
1.1	i-Coin project	5
2	SOA (SERVICE-ORIENTED ARCHITECTURE)	6
2.1	Benefits	7
3	I-COIN PILOT PLATFORM	9
3.1	The concept of i-Coin platform	9
3.2	Functionality	9
3.2.1	User authentication	10
3.2.2	Applications	10
3.3	Adoption	11
3.4	Development	13
3.5	Maintenance and benchmarking	14
4	REFERENCES	15



Sami Laaksonen

General Description
i-Coin

20.6.2007



1 INTRODUCTION

This document describes generally the concept of SOA (Service Oriented Architecture) and how it is related to i-Coin project. This document avoids talking about technical details.

1.1 i-Coin project

The main idea of the i-Coin (interregional Communication and Information Network) project is to develop and produce (design) a new online communication service for regional/local actors. Service will be developed and designed by the citizens in cooperation with regional/local actors in rural/structurally weaker areas.

i-Coin will enable people to initiate a dialogue and quickly obtain resolutions to their concerns using a communication system with terminals located at the countryside or over the internet. The overall objective of the project is to strengthen the capacity and confidence of local and regional administrations to provide better services and effectively engage in the adaptation to a knowledge society by developing solutions and policies on e-government.

The i-Coin project involves 10 partners from 7 countries around the Baltic Sea. The project has a budget of € 1.8 million from where € 1 million is a contribution from EU and the rest each partner stands for. The idea of the project was born in Svenljunga Municipality, Sweden when the broadband network was ready and it was possible for all villages to get a broadband connection. The focus in the project was to reach out to people who today cannot connect to the Internet or who are not even having access to a computer.



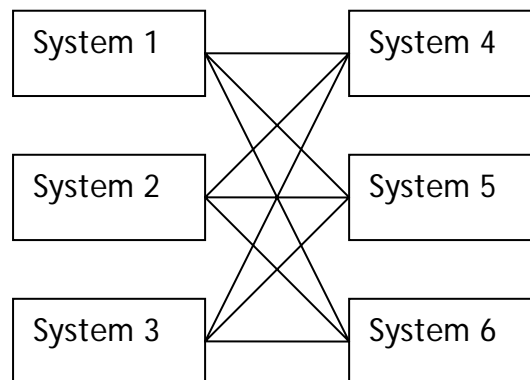
2 SOA (Service-Oriented Architecture)

SOA is an architectural style which goal is to achieve loose coupling among interacting software agents. Service orientation uses standard protocols and conventional interfaces—usually Web services—to facilitate access to business logic and information among diverse services. As the name suggests, SOA is based on services. SOA allows the underlying service capabilities and interfaces to be composed into processes. Each process is itself a service, one that now offers up a new, aggregated capability. Because each new process is exposed through a standardized interface, the underlying implementation of the individual service providers is free to change without impacting how the service is consumed.

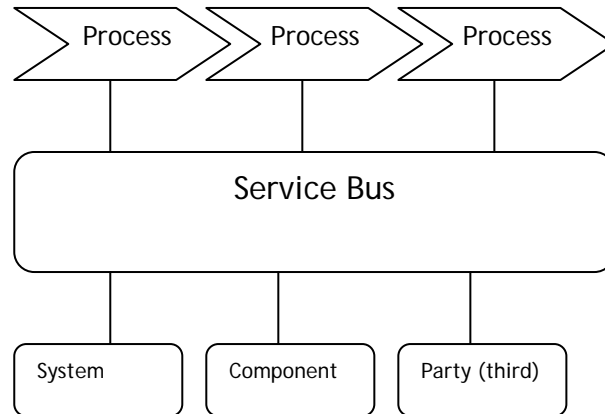
Example:

DVD: If you want to watch it, you put your DVD into a DVD player and the player plays it for you. The DVD player offers a DVD playing service. You can also play the same DVD in another DVD player.

Strong basic rule of loose coupling (a part of SOA philosophy) is causing massive improvements to e.g. system integrations. There is no need to build different kind of function or data connections between systems in SOA model (Picture 1 and Picture 2). Data flows go through service bus as a part of implemented business process and service requests themselves are building functional integration.



Picture 1: Directly connected systems



Picture 2: SOA model

Picture 1 and Picture 2 are illustrating changes when changing to SOA model. The most essential change is the process centrality. This will reflect not only to physical implementation but also to whole process of system work. The other central change is to move away from system centrality. This means in practise that part of the services can be produced in different systems and even in systems of third party.

Carrying on with the DVD example, the DVD movie is a component, DVD player is another component (or system), and the TV set is a third. Processes are a) putting the DVD in a player, b) Turning on the devices, and c) pressing the "play" button. As we all know, this works and the movie is shown on the TV screen.

Had the DVD process been created according to Picture 1, then, only those DVD movies created by Philips could be used in a Philips player. And only a TV set supported by Philips (and vice versa) could be attached to the DVD player.

2.1 Benefits

The main advantages of SOA-based systems are loose coupling of services, flexible modular development and maintenance, system and programming language independency, standard-based methods, and a much higher level of security compared to directly-connected systems.

Business Benefits of Service-Oriented Architecture:

- **Efficiency:** Transform business processes from replicated processes into highly leveraged, shared services that cost less to maintain.
- **Responsiveness:** Rapid adaptation and delivery of key business services to meet market demands for increased service levels to customers, employees and partners



- **Adaptability:** More effectively rollout changes throughout the business with minimal complexity and effort, saving time and money.
- **Scalability:** Add new servers if the traffic grows beyond your expectations. No need to shut down the service while you are updating the server. Just add a new server and then decide if you want to run two servers in parallel or switch off the older one.

IT Benefits of Service-Oriented Architecture:

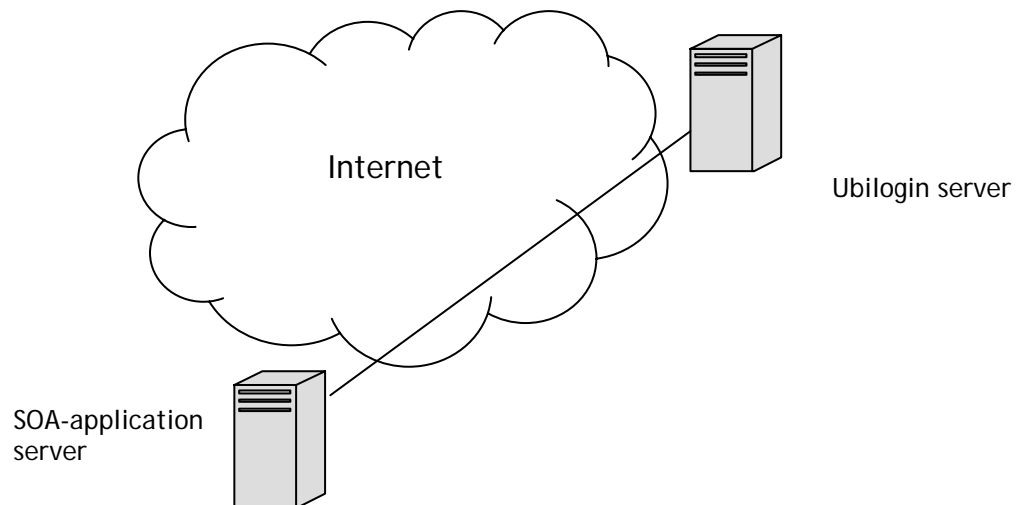
- **Reduce Complexity:** Standards-based compatibility versus point-to-point integration reduces complexity
- **Increased Reuse:** More Efficient application/project development and delivery through the reuse of shared services, previously development and deployed
- **Legacy Integration:** Legacy applications, leveraged as re-usable services, lowers the cost of maintenance integration
- **Better security handling:** Easier to manage which services are called and how the replies from other services are handled. No need for numerous ad-hoc "quick and dirty" connections to various databases and other outside systems.



3 I-COIN PILOT PLATFORM

3.1 The concept of i-Coin platform

The i-Coin platform is made up from two parts: The common platform, and the country-to-country varying local web based e-Government services. The common platform includes the SOA-application server and the Ubilogin server. The Ubilogin server is provided by Ubisecure and the SOA-application server is provided by this project and contains the SOA-application that works as the central component of the whole system.



Picture 4: The concept of the i-Coin platform

3.2 Functionality

The common platform's functionality is done in the background through the use of services. The common platform's main **VISIBLE** functionality for the user:

1. User clicks the link leading to the login-page.
2. User chooses the authentication method.
3. Types in a username and a password and presses the login-button (depends on selected authentication method).
4. If authentication is ok, a web-page containing user-information is displayed to the user.
5. When the user requests another web-page during the same session, rights are checked again and the service will be given, user rights allowing. This model will improve data security and minimize additional actions conducted by the user.

3.2.1 User authentication

User authentication is done by UbiLogin-module. UbiLogin is a single sign-on, authentication authorization solution for web applications. UbiLogin server and needed applications are provided by UbiSecure.

All users must be registered somewhere to use the authentication services provided by the UbiLogin server. They can be registered:

- In the UbiLogin server's database.
- In an external database (Active Directory, Ldap, Radius,...) maintained by the municipality or a company.
- In a large scale database, such as the banks or telecoms operators have on their customers. This can be used if the banks, telecoms operators - or other such organizations - are willing to sell their customer information for authentication purposes.
- In a country-wide database - typically maintained by the government authority or police - where personal information is stored and provided for authentication purposes.

When the user starts using a local e-Government web site and clicks on a link leading to "protected" content, he needs to be authenticated in the first place. UbiLogin will take care of the authentication of the user. If the user has more than one possible authentication method, he will need to select which method he wants to use. If not, authentication is a straight forward process.

After the authentication, the SOA application processes the user's request. It makes the connection to all databases and systems that can provide the information. The SOA application talks to the SOA layer, which is present in these databases and systems. If needed, the local SOA layer checks the user's rights to the information before delivering.

First the user has to sign in to the system and the UbiLogin-module provides the means to do this. After signing in the used website tries to use a service provided by the SOA-application. If the used service suggests that the user tries to access information that needs authentication, SOA-application sends another authorization request to UbiLogin-module. This functionality is hidden from the user and doesn't affect the experience of using the website. UbiLogin-module returns a message which contains information about whether the user can access the information or not. If the user has rights to access the requested database(s), the SOA-application may directly or indirectly through an interface fetch the desired data from one or more databases and send it back to the website for the user to see. In case the user doesn't have the rights to view the desired information, the SOA-application sends an empty or erroneous message back to the user. This way the user interface (website) can inform the user that he or she doesn't have the rights to access that information.

3.2.2 Applications

Applications suitable for the introductory architecture are mainly based on the concept of an Enterprise Service Bus (ESB) platform. An ESB is software (middleware) infrastructure that simplifies the integration and flexible reuse of business components using SOA. An ESB intermediates interactions between enterprise applications, business services, business components, and middleware



to integrate and automate business processes. Its main purpose is to support loose coupling by separating service consumers and service providers from each other. Most ESB-applications also provide tools for controlling and supervising application behavior. Normally an ESB-application also contains a service directory component which stores information about available services and works as a service database.

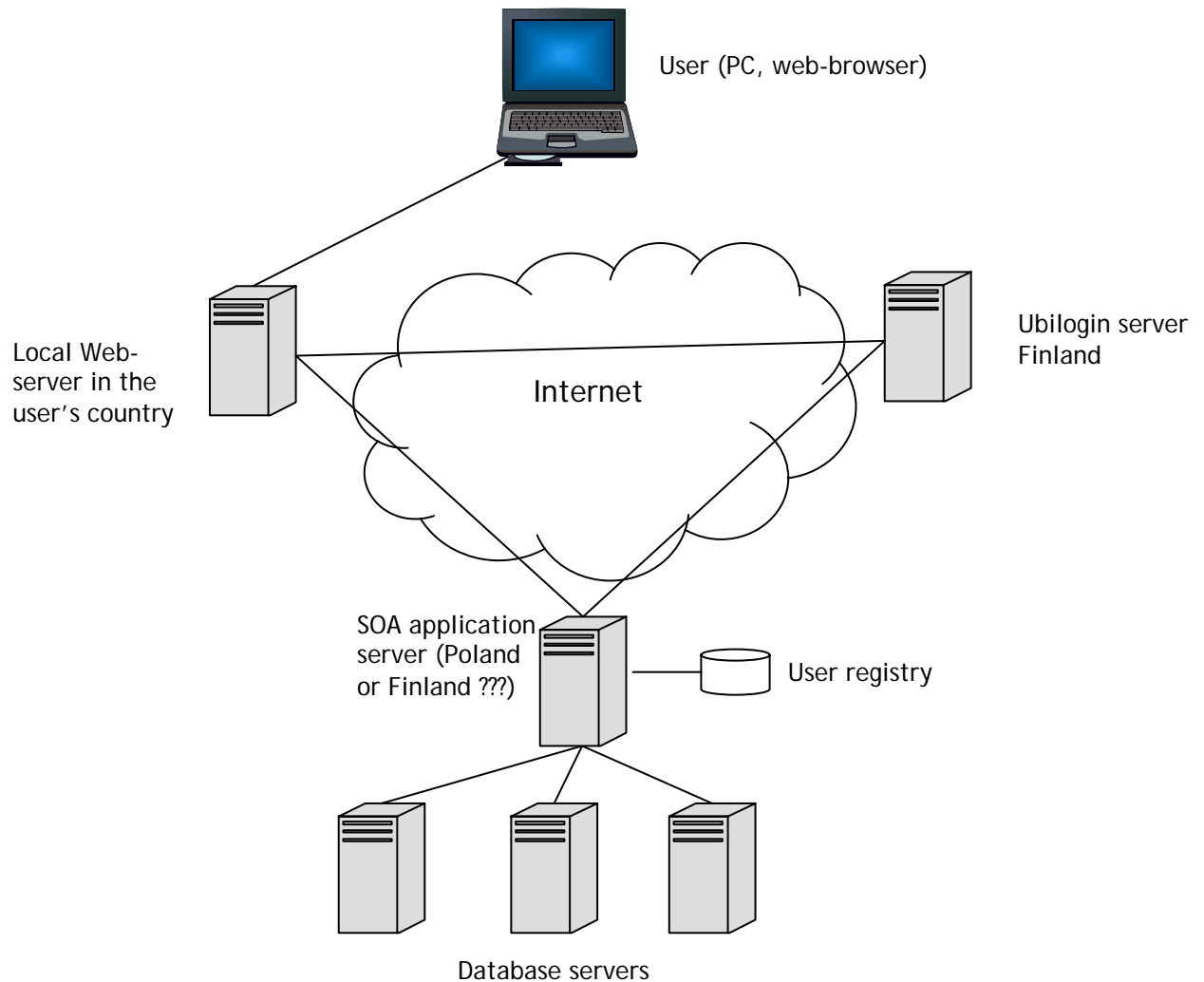
Open source software refers to computer software that has an open source code. This means that the software is available under some license that allows all the users to use, study, modify and redistribute the software and its source code whether its modified or not. Many open source applications are developed in a collaborative manner so that a number of developers all develop new features at will and publish them with the same open source license. Basically this means that open source software is free to use and develop. Nowadays there are both commercial and open-source applications that fulfill the basic requirements of an ESB.

The ESB-application to be used in this project is Mule ESB (<http://mule.codehaus.org/display/MULE/Home>). It's an open source product which can be freely developed and redistributed. It was chosen since it has great capabilities and good documentation compared to other competing open source products.

3.3 Adoption

Although the project is progressing continuously, clear steps for adoption of the system aren't available yet. Still, it is clear that it's possible to use the i-Coin platform by creating a link on the website for Ubilogin authentication and after this by invoking a single Web Service -interface. Basically the services are created in the SOA-application and they can be used straight from the website with a single click. Invoking this Web Service -interface requires some programming skills and a programming interface for a programming language that supports Web Services. The information about the interface needed to invoke the service will be delivered as the project proceeds.

Picture 4 below illustrates the physical architecture of the pilot system.



Picture 4: Physical architecture of pilot system

3.4 Components

User:

Basically the only application needed on user's computer is a web browser. A SOA-based system works just like any other website or web-application. The only thing users see is the website itself, all the functionality related to the SOA-application and the databases are hidden.

Web-application server:

The local web-server works just like any web server. It needs an operating system with server capabilities. Existing web servers may be used for this purpose. The only requirement for the web-server is that it must support "Web Services" -standard.

UbiLogin server:

UbiLogin server is provided by UbiSecure.

SOA-application server:

SOA-application server contains the SOA-application and an application platform which are tightly connected. The SOA-application is implemented using open source -software.

User registry:

User registry is a database for preserving user and user group information. In Finland we will use authentication services provided by major banks and no other user registry will be needed. How are the users to be registered and authenticated in your country?

Database servers:

Database servers are out of scope of the actual platform but are, of course, an essential part of the system. A database server has an operating system and a number of databases. If a database is directly connected to the SOA-application, so that the SOA-application may make direct queries to the database, the server doesn't necessarily need an application platform. Still, an application platform is needed in most cases.

3.5 Development

Component development in Mule ESB is handled with Java and XML. Java is used for implementing transformations of messages and basic functionality. XML, on the other hand, is used to control data flow in the ESB system. Mule ESB provides a development interface for an open source development platform called Eclipse. It provides both java and XML development interfaces. The software can be downloaded from <http://www.eclipse.org/>. Java and XML -implementations can also be produced with simple text-editors. In these cases non-graphical tools (apache's ant and maven) can be used to compile the produced Java-code.

The used introductory architecture enables a lot of different opportunities. At least the following advantages can be seen:

- The same SOA-application can be used by multiple different Web-applications through the service interfaces provided by the application. If a service implementation needs to be changed, it can be done in the SOA-application layer instead of making changes straight to all Web-applications. Basically this means less work and complexity and easier and more cost-effective development.
- New services may be added to the SOA-application and they can be found and used dynamically. This means that services don't have to be static. A user can for example search for a service that best suits his/her needs and then use it. This can be achieved as long as the user interface allows this kind of actions.



- System can be developed on a piece-by-piece basis by adding new services when they are finished. The system can be working even with only one service (which will probably be the case with the pilot-system).
- Services and user interfaces can be implemented with any programming language that supports Web Services. Standard-based interfaces and messages ensure communication between different kinds of systems. The logic in the SOA-application is implemented with Java and XML.

3.6 Maintenance and benchmarking

Mule ESB is controlled and configured with the help of several XML-files. These XML-files can be modified with the development interface provided for Eclipse or by using a basic text editor. Basically these files control the functionality of the software. Users and user groups are handled with the help of user registry.

Statistics information related to the use of the SOA-application, for example how many times a certain service has been used, can be handled through the use of a database or a simple xml-file connected to the SOA-application. Also some ready-made open source solutions are available for maintaining statistics. These ready-made solutions are tested and evaluated at a later time.

Mule ESB (SOA-application) doesn't currently have its own benchmarking application. Still, there are some ready-made solutions also for benchmarking that are compatible with Mule ESB. Benchmarking is mainly conducted by stressing the application - performing multiple tasks simultaneously - and measuring performance during the operations. A more accurate benchmarking plan will be created later on.



4 REFERENCES

1. SOA technicalities.doc
2. What Is Service-Oriented Architecture?
(<http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>)
3. SOA (<http://www.yliopistojenit.fi/weblehti/nro9/soa.html>)
4. The ABCs of SOA
(http://www.bea.com/framework.jsp?CNT=soa_abc.htm&FP=/content/solutions/soa/basics/)
5. Learn About Service-Oriented Architecture (SOA)
(<http://www.microsoft.com/biztalk/solutions/soa/overview.aspx>)
6. Mule ESB website
(<http://mule.codehaus.org/display/MULE/Home>)