Creating new university management software by methodologies of Service Oriented Architecture (SOA)

Tuomas Orama, Jaakko Rannila

Helsinki Metropolia University of Applied Sciences, Development manager, Bulevardi 31 PL 4000, tuomas.orama@metropolia.fi
Helsinki Metropolia University of Applied Sciences Project manager, Bulevardi 31 PL 4000, jaakko.rannila@metropolia.fi

Keywords
Service oriented architecture (SOA), Enterprise Service Bus (ESB), Curriculum management & planning, Course management & planning, Teacher work hour management & planning, Timetables and resource management & planning

1. INTRODUCTION

The number of IT systems in Higher Education Institutions (HEI's) is growing. This sets new challenges to HEI's IT management because the number of integrations between different systems are becoming more and more complex. Integrations made in a traditional way i.e. copying data from system to another lead to point-to-point integrations, which are difficult to manage.

This kind of design leads to a "spider web" of different kinds of integrations when implementing new systems to the IT environment. Eventually this kind of design usually leads to duplicated data when data is copied from database to another or loss of information integrity and quality when data is not synchronized correctly between different databases. When that happens there is master data problem and questions arise. Have different users already updated the same information in different databases? Which of many instances of information is true?

Different systems have usually very different user interfaces, which lead to problems in usability. Making new user interfaces to these systems can also be difficult because there can be business logic in the user interface layer.

Using different kind of protocols in integrations can also make problem solving more difficult. Integrations are made inside each system and there is no centralized monitoring of integrations.

The current systems architecture at Metropolia University of Applied Sciences (Metropolia for short) has grown quite organically and freely and is partly inherited from its predecessors. A clear need to benchmark systems architectures in other institutions of higher education both locally and globally existed.

A proof of concept on the use SOA methodology and tools in Metropolia was previously implemented and the internal processes and information flows were analyzed during this Proof of Concept (PoC). It became evident due to the nature of the several different information systems that a full-blown service bus was needed to enable systems to share data efficiently and in a coherent manner and to avoid single vendor lock-in situations.

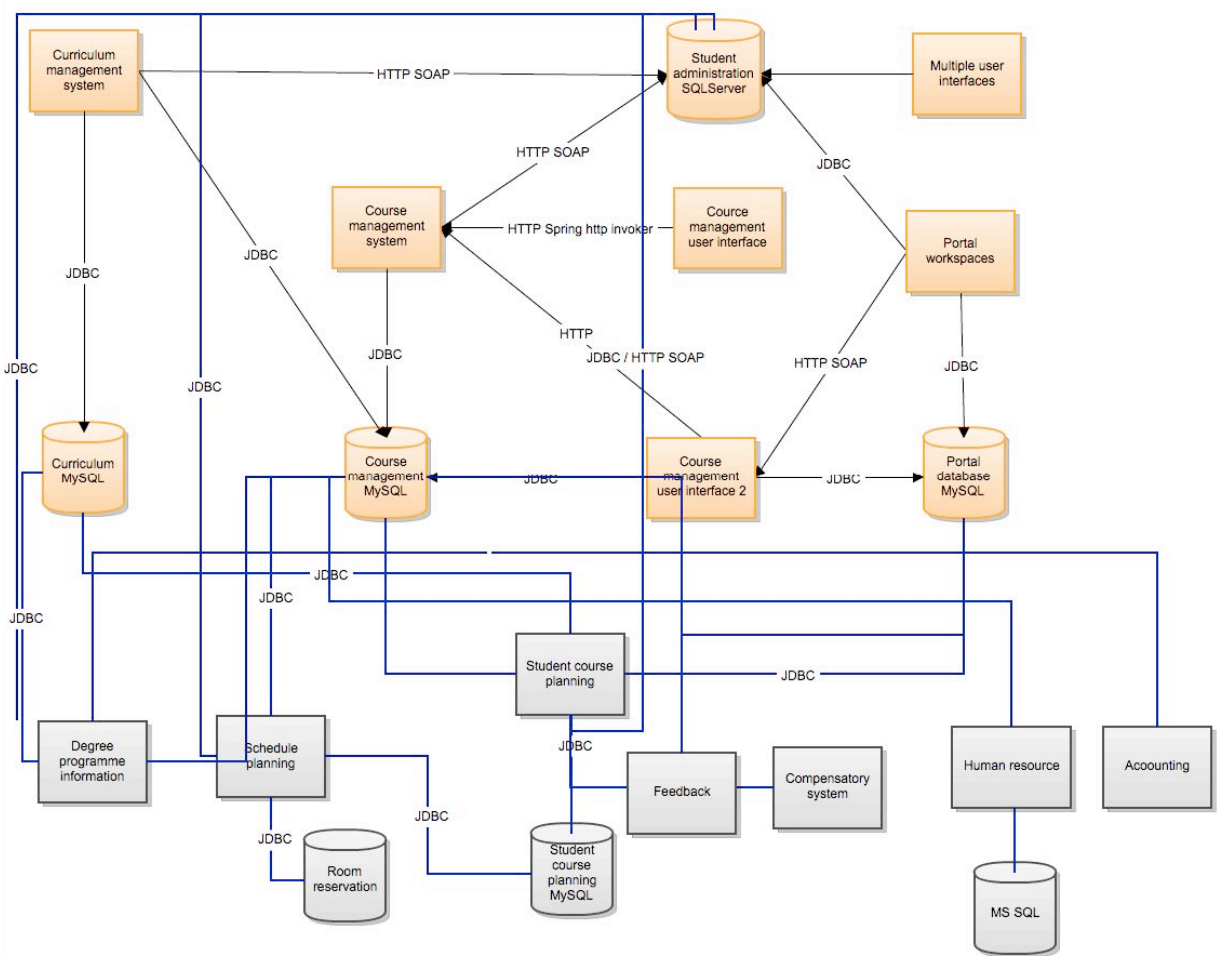

Figure 1: Current architecture presenting the "spider web" of integrations

Picture above presents current architecture where systems are integrated using different protocols and technology. Systems and integrations are usually implemented by different vendors or in-house. Integrations are point-to-point and they copy data between databases. This usually leads to a master data problem because of duplicated data.

Traditionally when building new systems the integrations between existing systems and the new system have minor priority. Common mistake is to think that the information created and maintained in the system is owned by that system and no other system would need that piece of information. Point-to-point integrations between existing systems are painful and prove that this theory is true. Systems have also grown larger and more complicated because of new needs for services must be implemented to those systems. This creates monolithic systems, which are almost impossible to maintain, develop or renew cost-efficiently.

Dependencies between the systems are also problematic. Creating point-to-point integrations, which are based on data synchronization make it hard to modernize and develop those systems independently of each other.

2. Different approach to the traditional architecture - Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) is not a standard. SOA is a set of principles and methodologies used in software designing and developing [1]. Because of this, creating competition document of creating software that is made in SOA way can be difficult task. This is because there is no just one way to do it. For example, you don't have to have an Enterprise Service bus to make SOA based software. SOA also doesn't define specific technologies or programming languages to do SOA.

Actually there is no tight binding to any programming language. This is very important aspect to understand when starting to design SOA software or buying new software that claims to be SOA. One aspect to understand is that SOA can be seen continuum from older concepts of distributed computing and modular programming [1]. One can say that SOA just defines the way of thinking and the design principles when creating software.

In service oriented architecture services are stateless and procedural. This approach is opposite to domain driven programming where objects are not stateless.

Metropolia adopted following design principles from SOA in the Peppi project:

1) Services are designed as reusable services,

2) Services can be composed to create new services,

3) Dependences between services are made as loose (loosely coupled) as possible,

4) Information is read and saved through service interfaces,

5) Services must be autonomous. Tight binding between services is not allowed.

6) Services are published using standard interfaces (e.g. SOAP Web service or REST).

7) Services statelessness.

8) User interfaces has no business logic

Other important things to take into the consideration are that changes made to the database won't affect the service interface or break user interfaces, which are connected to the service interface. In real world there will be a need to change the service interface at some point. When that happens one must ensure that user interfaces and other integrations won't be broken. Services must be versioned.

3. About the project

Peppi aims at becoming a comprehensive service-oriented system for curriculum planning and related e-services for teachers and it is the first piece of integrated system that will be implemented using the overall architecture presented in this overview documented. Peppi is a joint project of two large higher education institutions in Finland. To clarify things more: Peppi is a project where a new system for planners and teachers will be developed using the principles of the SOA based overall architecture, but at the same time it is a driver towards the full utilization of SOA in the organization. It is an internal learning process also, in addition to being a software development project.

The project is a complete renovation of the application architecture, user interfaces and design methods. The project has started in 2010 and all the services made in the project will be in production by the end of 2013.

Table 1: Project in numbers

Project members	Budget (2010 - 2013)	Documentation so far (in pages)	Estimation of man-days used in the project	Estimation of code lines created in the project
87	2,7 million €	600	5500	150 000

Planning tools include curriculum/syllabus planning, rule engine, course planning, course instance planning, organization planning, expense planning and budgeting, teacher work hour planning, schedule planning. Services for teachers include virtual learning environment, awarding process handling, communication tools, own schedule handling, own workload handling etc. Services mentioned above are made by the end of 2012. Some of the services like curriculum/syllabus planning are already made, tested and ready for production.

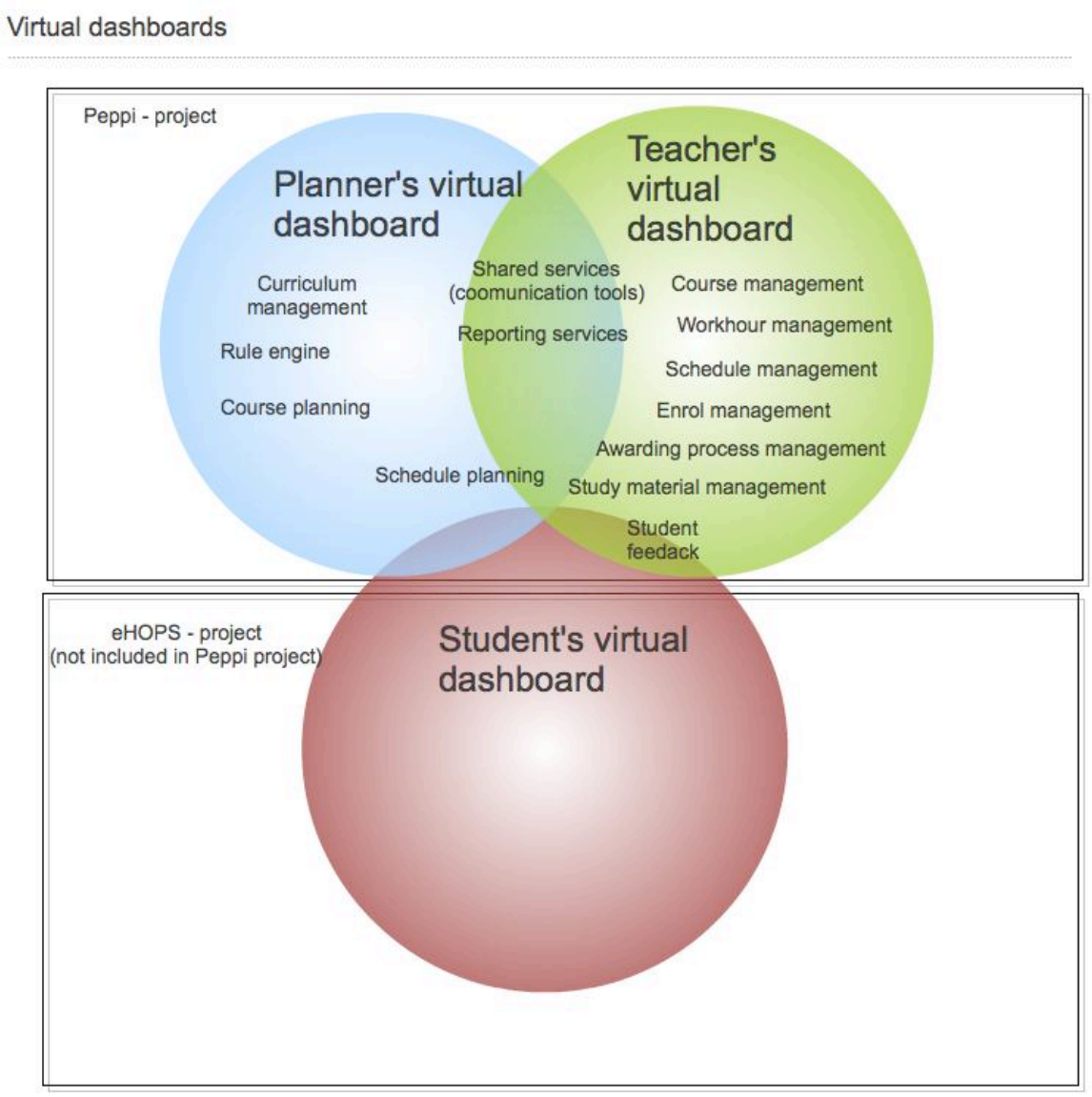

Figure 2: Virtual dashboard and their services

Picture above presents dashboards in Peppi. Peppi project produces two virtual dashboards: Planner's dashboard and Teacher's dashboard. Services are assembled by process-based and user-oriented way in the dashboards. This approach improves user experience because functions in system are logically arranged and presented at the right time for the user.

4. PEPPI ARCHITECTURE

Architecture

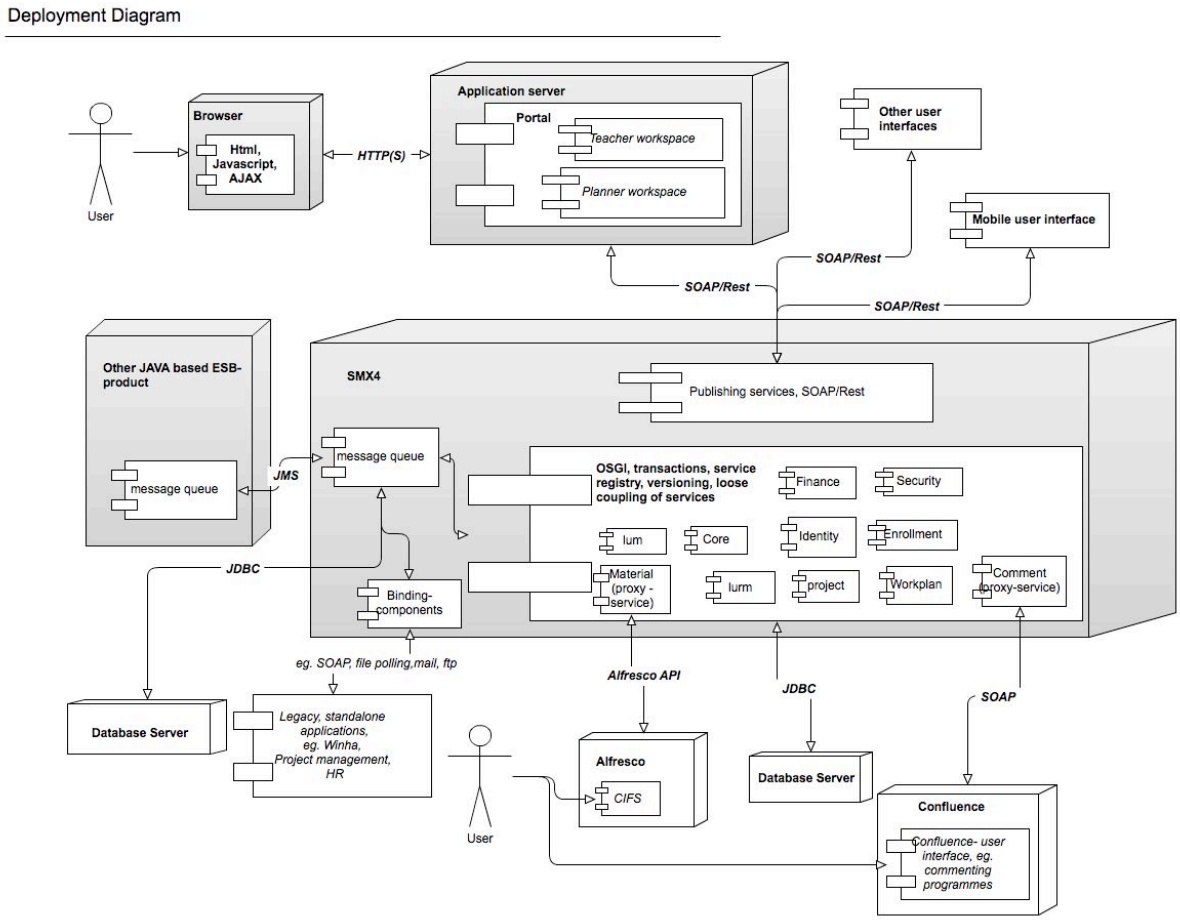

Figure 3: Deployment diagram in Peppi

In figure 3 is presented Peppi Deployment diagram. In the core of deployment is SOA platform (ServiceMix). ServiceMix is runtime environment for different kind of services. It also has support for Enterprise integration patterns (EIP). Service interfaces can be published for different protocols and standards. Services in Peppi are implemented with OSGi technology which interfaces are published as SOAP Web services and REST. ServiceMix contains also message queue to send and receive messages to 3rd party systems. Message queue can be configured to persist messages during message exchange. This improves fault tolerance and improves robust architecture compared to present integrations.

Peppi software will be implemented in J2EE. The main reason for choosing Java was the fact that many of the existing systems had been implemented in Java over the years and there was already a lot of in-production experience.

When using OSGI-technology and making services as OSGI bundle, which are run in ServiceMix4 it is possible to run different versions of service interfaces simultaneously. This provides a significant additional value over normal J2EE behavior.

User interfaces

User interfaces are built into one centralized portal for teachers and study program administrators & planners. User interfaces take advantage of rich interfaces like HTML5, CSS3 and AJAX. In Peppi there is a clear and precise separation of user interfaces from all business logic and a very lightweight approach to producing UI components is taken into use.

User interfaces are made such way that they do not include business logic. All the logic is in the service. Noticeable is also that everything you do in user interface can also be done by programmatically in service interface by SOAP or REST.

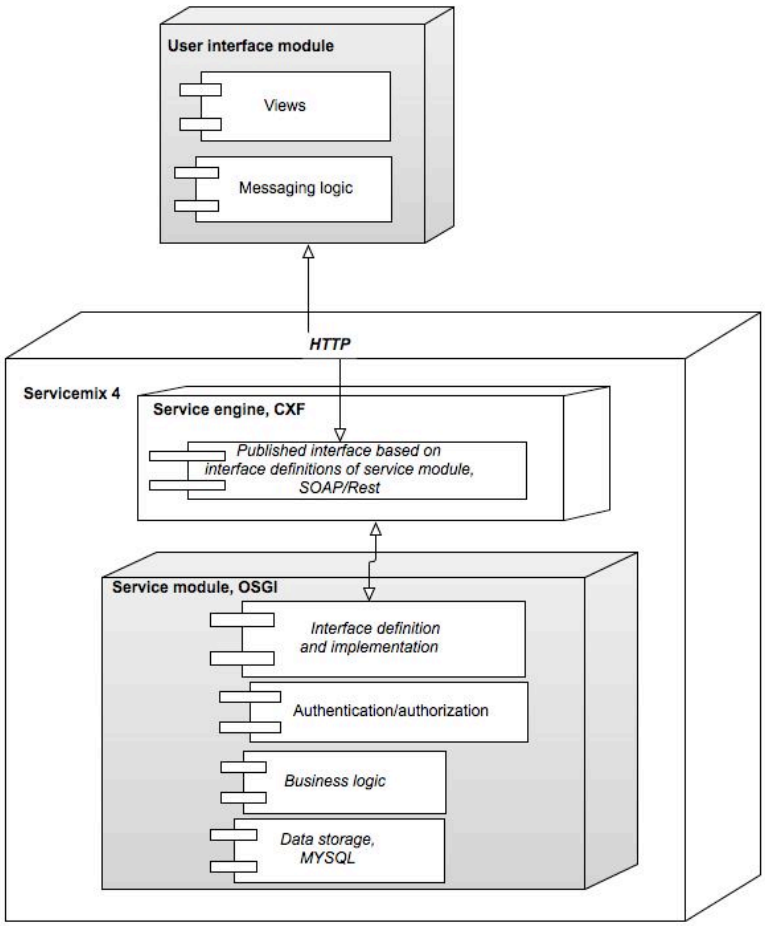

Figure 4: Service modules and user interface modules

Picture above describes how user interfaces are connected to service interfaces and how service interfaces are connected to OSGi service modules.

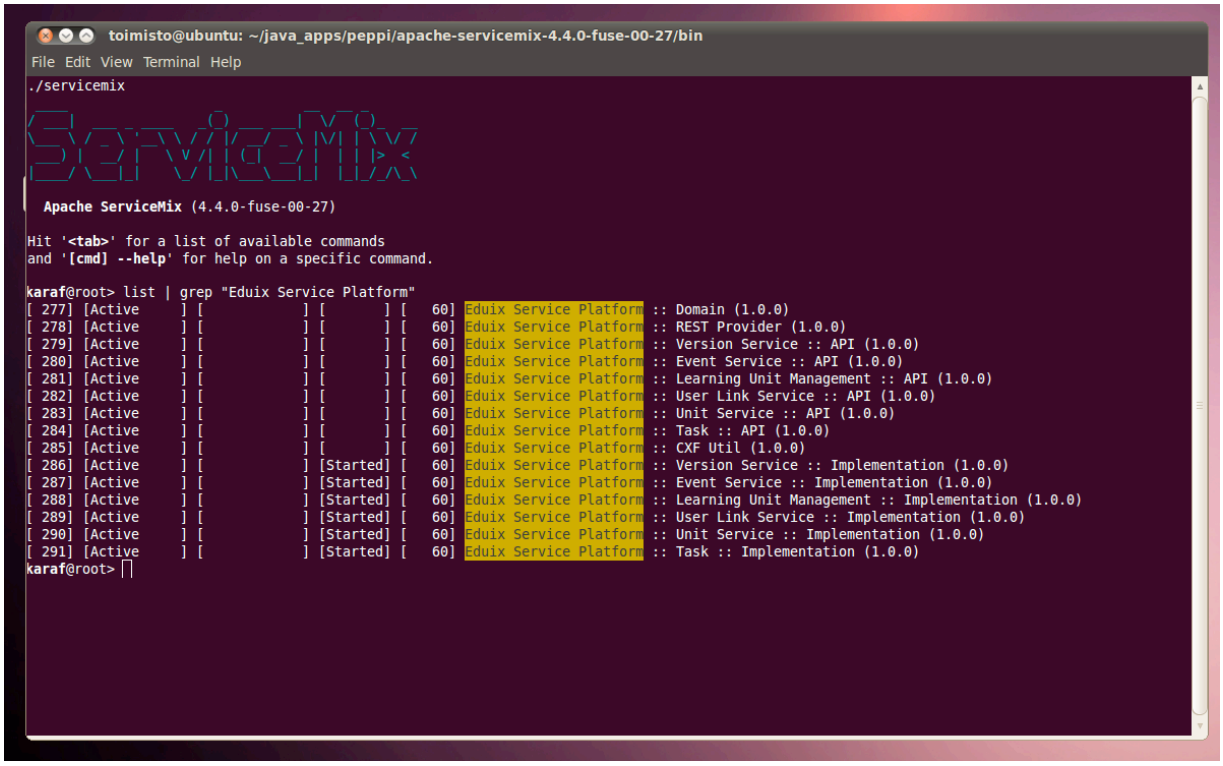

Figure 5: Services running in Servicemix OSGi container

In the picture above you can see modules containing interfaces (API) that are separated from implementations. Services are autonomous and there are no tight bindings between them. Modules are changeable. This is one of the principles in SOA.

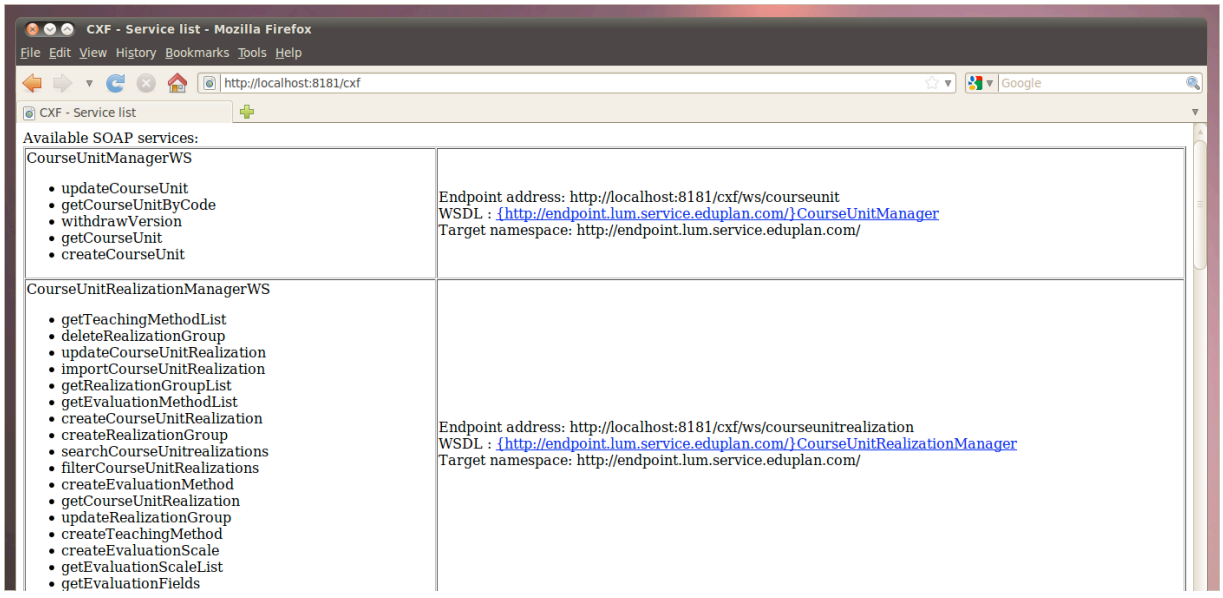

Figure 6: List of service interfaces in Apache CXF

In the picture is a list of published service endpoint interfaces with WSDL descriptions. Interfaces are platform independent and are based on standards (SOAP based Web services). User Interfaces are connected with SOAP interfaces. Endpoint interfaces are reusable to different kind of clients (user interfaces, systems integrations e.g.).

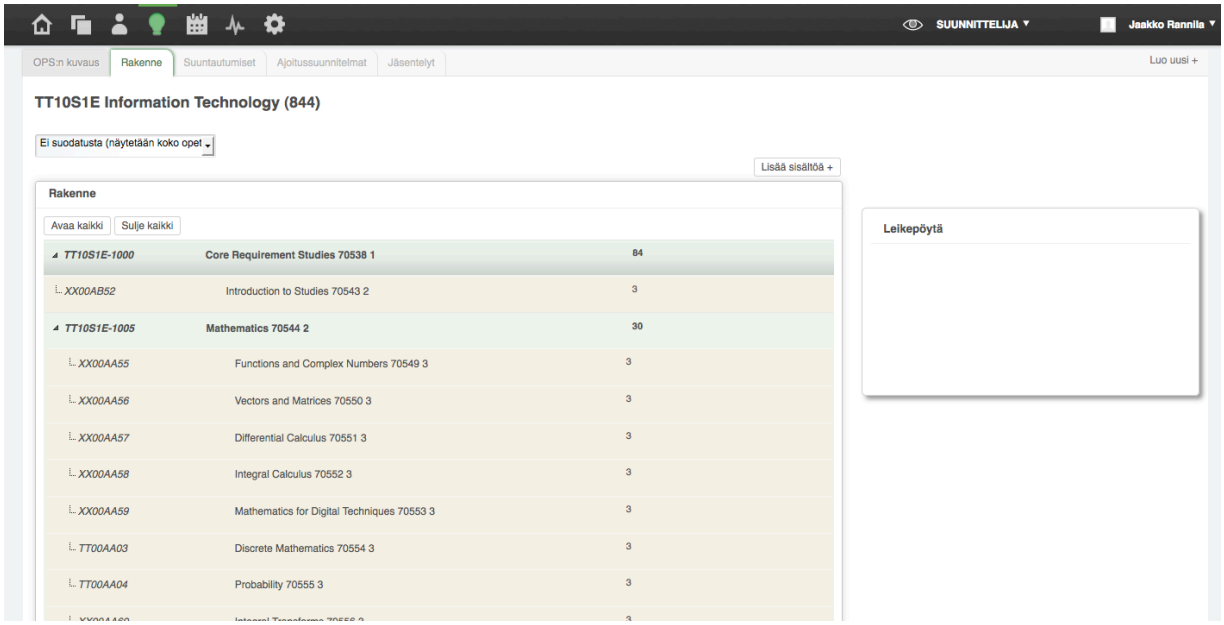

Figure 7: Snapshot from user interface of planner's virtual dashboard; Curriculum management

In the picture above you can see figure 6 decorated with graphical components. Interfaces can be used programmatically (Figure 6) or from graphical user interface (Figure 7). Graphical user interfaces use rich interfaces such as CSS3, JavaScript and HTML5. Graphical interface has multilingual support and they are made using standards like JSR168 and JSR286.

Portlets are implemented as light as possible so they can be aggregated to different kind of views to improve user experience. Portlets doesn't contain any integration logic or business logic. This makes portlets renewal easier and improves graphical user interface performance and reliability.

This kind of architecture allows making new interfaces i.e. mobile phones without making new software. This solution supports one the SOA principles of making services that are reusable.

Using standards and open source software

All the services are made by using standards like Opens Services Gateway initiative framework (OSGi), Java Business Integration (JBI), Web Service and Portlets (JSR168 and JSR286). Services use Enterprise Service Bus (ESB), which is Apache ServiceMix. All of services are published through Apache CXF by SOAP interfaces and REST/xml, REST/json interfaces.

Using open source software in Peppi project we can achieve significant cost savings and supplier independency. Furthermore using standards in every solution we can achieve solutions that have a very long lifespan.

Servers and clustering for high availability

Both Liferay and Servicemix4 can be clustered. Clustering ServiceMix can help to achieve high availability. Updates to the services can be made without service outage, because updates can be made one cluster at the time.

Benefits that can be achieved

There are several benefits from the design methods we have presented here. In Peppi testing environment there is organization data, curriculums, courses and course instances of two separate universities. Because of the technology and data model in Peppi, it can easily be driven as a cloud service. There is no limit to add more organizations for using it as a cloud service. All the components are scalable.

With the help of ESB it is very easy to make new service interfaces to the environment. It is also possible to make on-the-fly schema changes inside ESB. This possibility gives us an opportunity to deal with any information exchange with other HEI's.

5. REFERENCES

[1] Wikipedia (2012). Service-oriented architecture, from: http://en.wikipedia.org/wiki/Service-oriented_architecture

[2] CSC (2012). Raketti-Wiki, from: https://rakettiwiki.csc.fi/wiki/XDW/WebHome

[3] Ministery of finance (2012). Perustietovarantojen käytön kehittäminen, http://www.vm.fi/vm/fi/16_ict_toiminta/01_yhteentoimivuus/index.jsp