

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computers



Bachelor thesis

Student's church II – plug-in support

Marek Sacha

Supervisor: Ing. Jiří Chludil

Study programme: Software Engineering and Management

Specialization: Software Engineering

9th June 2009

Non est ad astra mollis e terris via

Declaration

I declare I disposed this work on my own and I used only the references mentioned in the attached list.

I do not have any serious reason against usage of this school work in the sense of §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

9th June 2009

Prague

Marek Sacha

Acknowledgements

I would like to thank to my supervisor Ing. Jiří Chludil, for his advice and support, RNDr. Bohuslav Škop, CSc, for technical and time support, Tereza Skopalíková, for proof reading, all the colleagues from Student's church and Mantichora projects, for successful cooperation, and finally Ing. Miroslava Sachová, for the vision.

1. Setting

Implement module of web-oriented information system Student's church (bachelor thesis 2007) that allows simple extensibility for new functional modules.

1. Select appropriate technology of the solution
2. Demonstrate functionality of your solution in Student's church and Mantichora applications
3. Prepare tutorials for usage of your technology

2. Abstract

The purpose of this work was to provide Student's church and Mantichora projects with a plug-in support and also a possibility of system integration (also into other information systems). This bachelor thesis compared many different technologies. **Web services** (SOAP, WSDL) – an existing industry standard – were chosen as the appropriate solution of the task and consequently implemented.

This work brought functional, stable and safe solution for inter-component communication. Created code was reviewed and tested in real applications also by other members of development teams. The developers were provided with simple tutorials and examples of particular implementation.

2. Abstrakt

Cílem této práce bylo upravit systémy Studentova berlička a Mantichora o podporu rozšíření pomocí pluginů a případně i o možnost systémové integrace (i do dalších informačních systémů). V bakalářské práci je porovnáno mnoho různých technologií. Jako nejlepší řešení byly vybrány a implementovány tzv. „**Web services**“ (SOAP, WSDL) – existující průmyslový standard.

Tato práce přinesla funkční, stabilní a bezpečné řešení pro komunikaci mezi jednotlivými komponentami. Vytvořený kód byl překontrolován a testován v reálných aplikacích dalšími vývojáři jednotlivých komponent. Pro potřeby vývojářů jsou přiloženy jednoduché tutoriály a příklady pro konkrétní implementaci.

3. Contents

- 1. Setting..... 1
- 2. Abstract..... 3
- 2. Abstrakt..... 3
- 3. Contents..... 5
- 4. Introduction..... 7
 - 4.1 Student’s church project..... 7
 - 4.2 Goal..... 7
 - 4.2.1 Main goal 8
 - 4.3 Mantichora 9
 - 4.3.1 Main goal 9
- 5. Background research..... 11
 - 5.1 Architecture..... 11
 - 5.1.1 Architectures comparison 11
 - 5.2 SOA (Service Oriented Architecture)..... 13
 - 5.2.1 Implementation of SOA..... 14
 - 5.3 Web Services 16
 - 5.4 SOAP 16
 - 5.4.1 WSDL..... 17
 - 5.4.2 UDDI..... 17
 - 5.5 Mantichora real time communication channel..... 17
- 6. Analysis..... 18
 - 6.1 Student’s church 18
 - 6.1.1 Requirements..... 18
 - 6.1.2 Deployment model..... 20
 - 6.1.3 Use cases..... 21
 - 6.2 Mantichora 24

6.2.1	Requirements	25
6.2.2	Deployment model	25
6.2.3	Use-cases	26
7.	Implementation	29
7.1	Class Model	29
7.2	Implementation environments	31
7.2.1	PHP	31
7.2.2	Java	33
7.3	Testing	35
7.3.1	Unit testing.....	35
7.3.2	Architecture & SOAP design testing.....	35
7.3.3	Results of informal reviews.....	36
8.	Security.....	37
8.1	Data protection	37
8.2	Invisible security barrier	38
8.2.1	Client - Kernel.....	38
8.2.2	Plugin – Kernel.....	38
8.3	Cross site scripting.....	39
8.4	Web services and security features	40
9.	Conclusion.....	41
9.1	Future progress	41
9.1.1	Project management.....	42
9.1.2	Student’s church.....	43
9.1.3	Mantichora.....	43
10.	Bibliography	45
	Appendix A.....	i
	Attached CD.....	i

4. Introduction

4.1 Student's church project

Student's church is a 4-year old running software project focused on providing a communication solution for students and other academic staff (professors, lecturers). The main purpose was to offer simplification and unification of multiple other solutions used by each individual or study group.

Firstly, the system enables students to see their progress in each course, to see important milestones during semester and to communicate with their lecturer. Secondly, the system enables lecturers to store various necessary types of data – attendance, testing results or semestral projects and much more.

“Proposal of solution ensues from multi-annual survey between students and from need for simplifying work on side of pedagogues.” (1)

4.2 Goal

The purpose of this work is to provide developers of Student's church and Mantichora projects with validated and easy way of system integration (in terms of integration of plugins and other third party tools) to one solid application. The majority of this work is focused to provide **analysis of the solution** based on a choice of existing technologies or creation of a new one; secondly **basic implementation** must also be given to the developers as well as manuals and developer's packages for fast development.

Feedback

There has been a wide feedback based on practical usage of Student's church in last 2 years. The feedback could be divided into three main points.

1. Security
2. Extensibility
3. Possibility of customizable user interface and customizable functionality

Each area of these three brought many aspects which had to be considered or directly covered by this work. Majority of those aspects is summarized in the following table.

Area	Aspect
Security	Closed source kernel Encrypted communication User identity and validation Safe user interface transfer
Extensibility	Need for additional functionality Possibility of team development Platform independency Needed support for technological boost
Possibility of customization	Personal user interface Personal data model of stored data Mobile devices access Data transformations

Each of mentioned aspects will be discussed, explained and solved in the analytical part of this work.

4.2.1 Main goal

With the knowledge of the feedback collected it is possible to specify main goal of the work according to given task. This project must develop a solution, which will provide Student's church team with plugin architecture that:

- **is platform independent**
- **covers security measures**
- **is easy to implement in particular plugin or library**
- **provide developers with debugging options**
- **is well documented**
- **provides developers with tutorials and help**

All these features could be considered as important milestones, which could be easily measured.

4.3 Mantichora

Mantichora is the second team project this work was supposed to solve some issues of communication infrastructure. Mantichora is a new team project which is focused on simulation. Mantichora itself consists of three main modules – physical engine, visual engine and communication engine. These main parts will be supported by various editors (editor of scene, editor of physics).

Networking issues are mainly connected to cooperation of physical engine with visual engine, which are fully independent. The main difference between Student's church and Mantichora is simple requirement for real time and fast communication channel of mentioned modules. Quite different solutions could be expected.

Other networking issue of Mantichora modules is independent **network data storage**. In this case the architecture of solution is in contrary to real time communication channel quite similar to problems in Student's church architecture. This work will be focused mainly on this issue.

Last issue is connected to negotiation of communication channels, configuration and user profile. All three points may be covered by usage of appropriate network communication technology allowing independent services to automatically negotiate shared configuration and communication and it would also allow user to interact and set own preferences.

4.3.1 Main goal

- Analyze and design functionality and interface of independent **network file storage**
- Analyze and design functionality of **negotiating communication and configuration**

5. Background research

It was necessary to do a background research in the beginning of the whole work to be able to consider any existing solution or industry standard. This problem appeared to be quite often. There are several possible solutions available and also an option of creating own technology with custom features.

5.1 Architecture

The most abstract task considering the solution is the choice of the architecture. There are many different well-described and documented software architectures. Developing a new one would be a very complex task of a high risk. Work is therefore focused only on comparison of existing solutions.

5.1.1 Architectures comparison

There are four main requirements important for the choice of the architecture based on the main goal.

Independency

Independency is used in the terms of platform and hardware independency. It is important to invent a solution which could consist of different parts written in different environments and running on different operating systems – for instance test evaluation software written in java (including CRC module) or software for evaluating of authenticity of semestral projects written in Matlab.

Access control

Access control is crucial for a model of user rights in the whole project. It is considered how easily the model of user rights could be implemented and also whether there is any fundamental support in the architecture.

Scalability

Scalability is important in the means of running on different physical machines and also balancing of some computing time or memory limit sensitive tasks (e.g. authenticity evaluation).

Security

Securing access to the core and the communication with extensions is another crucial part. **Student's church** contains personal data (e.g. test results) of many students which must be protected even because of the Czech law¹. Architectures are evaluated according to possibility of securing the model of the architecture.

Comparison of different architectures

Architecture	Independency	Access Control	Scalability	Security issues
Monolithic (2)	None	One solution for whole application	Limited (threads)	
Software componentry (3)	None	One solution for whole application	Limited (threads)	
Three-tier model (4)	None	Tier-based	Tier-based	Tier-based
Plugin architecture (5)	Limited	Plugin-interface based	Limited (threads)	Plugin-interface based
Peer to peer (6)	Full	Peer-pair based	Full	No single point of failure, replication of data
Service oriented architecture (7)	Full	Service-based	Full	Service based security trade-off (availability of services, access parameters)

¹ Zákon č. 101/2000 Sb., o ochraně osobních údajů (účinné znění) (27)

Service oriented architecture provides developer with majority of requested features. **Monolithic** and **software componentry** architectures simply do not provide easy-extensibility which results from their fundamental model. There were three possible options – **plugin architecture**, **peer to peer** and **service oriented architecture**. Plugin architecture is too tightly connected to software interfaces and also its “independence” is limited (e.g. Mozilla Firefox plugins). **Peer to peer** architecture has many necessary features and in fact offers a technology with no single point of failure. On the other hand this feature results in decentralized communication model and data replication on each peer which is unacceptable from security reasons and solid database management.

These results in the choice of **SOA** (Service Oriented Architecture) – services are fundamentally independent. Each service could implement its own unique access control policy as well as one service could represent access control policy for whole system. Independency of services is tightly connected to powerful scalability. Security measures are mentioned in the architecture description. Each service could trade-off its security policy and could require different access prerequisites (e.g. type of cipher).

5.2 SOA (Service Oriented Architecture)

Service Oriented Architecture is based on following principles (7):

- *“Reuse, granularity, modularity, composability, componentization and interoperability”*
- *“Standards compliance (both common and industry-specific)”*
- *“Services identification and categorization, provisioning and delivery, and monitoring and tracking”*

Another important principle which appears in SOA model is principle of **loose coupling** (8) which is particularly important to Student’s church project. Loose coupling stands for a link between two services which are communicating together but also have a big level of independence. This could be done in different ways:

- Presence of unified messaging middleware
- Specific format using middleware to do data transformations

All of this leads to data model independence and also working independency – failure or unavailability of one service does not affect other one. Finally, all of these characteristics also imply that SOA brings possibility of fast changes in each module which, in fact, do not affect communication middleware and do not imply changes in other modules. For example credit card payment provider could completely rewrite payment processing and its backup system and in the same time interface and code of clients do not change at all.

5.2.1 Implementation of SOA

Nowadays, there is a wide range of possible implementations of SOA including SOAP, RPC, DCOM, CORBA, Web Services or WCF. Next task was to choose the best alternative from mentioned implementations.

There are also multiple evaluation methodologies of these implementations particularly for corporate usage², but in this work are not fully acceptable because of economical factors which are hard to measure in university environment.

SOAP

- Simple Object Access Protocol (9)
 - Message exchange protocol based on XML
 - Client-server model
 - Open industry standard
 - Part of implementation of Web Services

RPC

- Remote Procedure Call (10)
 - Client-server model
 - Subroutines executed in another address space
 - Interface description language – unified description of interface to allow being used on multiple platforms, languages
 - Various implementations (e.g. XML-RPC, Java RMI)
 - Predecessor of SOAP

DCOM

- Distributed Component Object Model (11)
 - Proprietary Microsoft technology
 - Communication of components across network

² 10 Measures for Successful SOA Implementations (28)

infrastructure

CORBA

- Common Object Request Broker Architecture (12)
 - Open industry standard (Object Management Group)
 - Strong typing
 - Data transfers in binary form (compression)

Web Services

- “A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.”³
- Includes SOAP and other standards
- WSDL 2.0 is considered to be a good implementation of RESTFull Web Service⁴

WCF

- Windows Communication Foundation
 - Proprietary protocol
 - Part of .NET Framework

The technology finally used for Student’s church was decided to be Web Services (SOAP, WSDL ...) because of multiple reasons.

- Web services are platform independent
- Web services are language independent
 - There are multiple existing implementations in various programming languages (including Java and PHP)
 - Fast development of services
 - Self-documentable structure of XML (both SOAP message and WSDL – service description) – reduced time costs and resources costs
- Web services are the industry standard
 - SOAP became a W3C Recommendation 24. June 2003 (13)
 - Market leaders are actively using SOAP (Microsoft, Sun, Google)
- SOAP communicates over standard protocols (e.g. HTTP, HTTPS)

³ Web Services Glossary (29)

⁴ Representational State Transfer (30)

- This feature covers problems with proxies and firewalls in institutional network infrastructure in contrary to RPC or DCOM
- WSDL 2.0 is considered to be a good implementation of RESTFull Web Service
 - REST (Representational State Transfer) is communicational architecture based on the following principles: client-server, stateless, cacheable, layered (14)
 - E.g. World Wide Web is an ultimate example of RESTFull design
 - By Roy Fielding – one of the principal authors of the HTTP specification

5.3 Web Services

Web services are independent application components, which communicate over standard open protocols and use self-documenting interfaces. Web services could be discovered automatically and provide simple access to variety of complex functionality bringing reusability and system integration.

Web services contain three main parts:

- **SOAP** – Simple Object Access Protocol
- **WSDL** – Web Service Description Language
- **UDDI** - Universal Description, Discovery and Integration

Good tutorials and comprehensive description could be found in several resources (15), (16). The purpose of this work is to show practical usage in projects Student's church and Mantichora. Because of that, next paragraphs bring only brief description of 3 pillars of web services.

5.4 SOAP

Simple Object Access Protocol is a standard for exchange of structured content in a form of XML messages



Figure 1: SOAP call schema (31)

In the picture there is a simple schema of communication using SOAP as messaging protocol – both request and response are encoded using SOAP. The communication itself is stateless.

SOAP is originally described on w3.org (17).

5.4.1 WSDL

Web Service Description Language is a tool (XML language) for describing a web service. In the format there are also used other industry standards like XML Schema (definition of custom data types).

WSDL provides web services with a strong tool for description which is self-documenting and readable for both developer and robot. WSDL actually makes possible “loose bind” between particular web services.

WSDL is originally described on w3.org (18).

5.4.2 UDDI

Universal Description, Discovery and Integration is a kind of complex database providing clients with information about registered web services. Because of its complexity and character of whole project (not a complex business solution) integration of Student’s church or Mantichora into UDDI was not analyzed and also not realized.

UDDI is originally described on oasis-open.org (19).

5.5 Mantichora real time communication channel

There is required a high performance real time communication channel for streaming results of physical engine to visual engine over the network in Mantichora. Web services use middle layer (SOAP) that is characteristic by large overhead (XML). Until there will be an improvement of SOAP introducing binary XML format which would reduce amount of overhead (particularly for smaller messages), communication over SOAP would be impossible because of high latency. Also TCP packets used by HTTP protocol are worse for streaming (latency of constructing a channel) than doing the same in UDP.

6. Analysis

The purpose of this part is to bring comprehensive view of solved problems and invented design of solution. According to goal of the project mentioned in the introduction all important milestones will be discussed and solved.

6.1 Student's church

A lot of information based on previous school projects, bachelor thesis by Jiří Hunka (1) and practical usage in university environment brought very detailed view of whole project both its advantages and disadvantages and requirements for the further functionality.

6.1.1 Requirements

Platform and language independent extensibility

The original requirement based directly on setting is a platform and language independent extensibility.

At first, independency requirement comes from need of support for different programming languages and programming environments. The complex tasks which could **Student's church** be used for in the future are very different (wide range of services starting at automated publishing of semestral projects across complex work authenticity evaluation finishing with statistical analyses) – for each different situation there are new factors influencing choice of platform, language... . This could be caused by experience and focus of students, need for different approach (low level programming bringing excellent performance, artificial intelligence algorithms) and last but not least by hardware availability.

Consequently, extensibility requirement is closely connected to previous one. Extensibility allows single authors or groups to work simultaneously and independently on separate projects. Development of Student's church and its extensions would be faster, smoother and could be done with lower costs (time, number of developers). Functionality of extensions would not affect the kernel.

Team development

For future development of **Student's church** team cooperation is a crucial factor. In the past the situation was directly opposite. The kernel of application was closed source because of the security issues. This resulted in extremely low number of people with an access to the system architecture and implementation and eliminated any possibilities of team development or cooperation. Limited features, absence of security audit and code review by third party and development costs (particularly time) were simple results.

Close to the team development is also the requirement of simultaneous development. Parallelization of developed projects connected to Student's church would offer possibility of boost of its functionality and metaphorically added value.

Customizable data model

Customizable data model would allow administrators to construct specific models of semesters, tests or students' profiles. For instance, requirements for successful completion of a subject change almost in each particular case. This requirement appeared almost in the very beginning. However this requirement is impossible to accomplish just by plugin architecture and require fundamental changes in Student's church kernel and also appropriate plugin infrastructure.

Customizable user interface

Customizable user interface would allow users to modify environment with focus on style of work and effectiveness. As important as style of work is also device, which is used as user-interface to the system. Support of various mobile devices could be easily built-in.

Security issues

Student's church itself contains variety of personal and sensitive data of students. Its absolute necessity to offer appropriate security politics and also user access politics. Milestone of security issues involves several topics e.g. code review, penetration tests, cryptography and various others. The communication model must pay attention to this requirement very much.

Source code availability

This requirement is closely connected to the team development. Students are able to continue with development of various previously finished parts if the source codes are available and well documented.

6.1.2 Deployment model

For better perception of whole problem and modeling of communication architecture it is necessary to construct a deployment model of whole information system. Than it is possible to construct and solve particular use cases requested by other team members.

There is an example of possible deployment situation. Student's church kernel is presented in current conditions with usage of MySQL database. Two possible plugins (there could be almost unlimited number of various others) with different execution environments and two different clients (PC and mobile device) show ordinary deployment.

Student's church deployment model

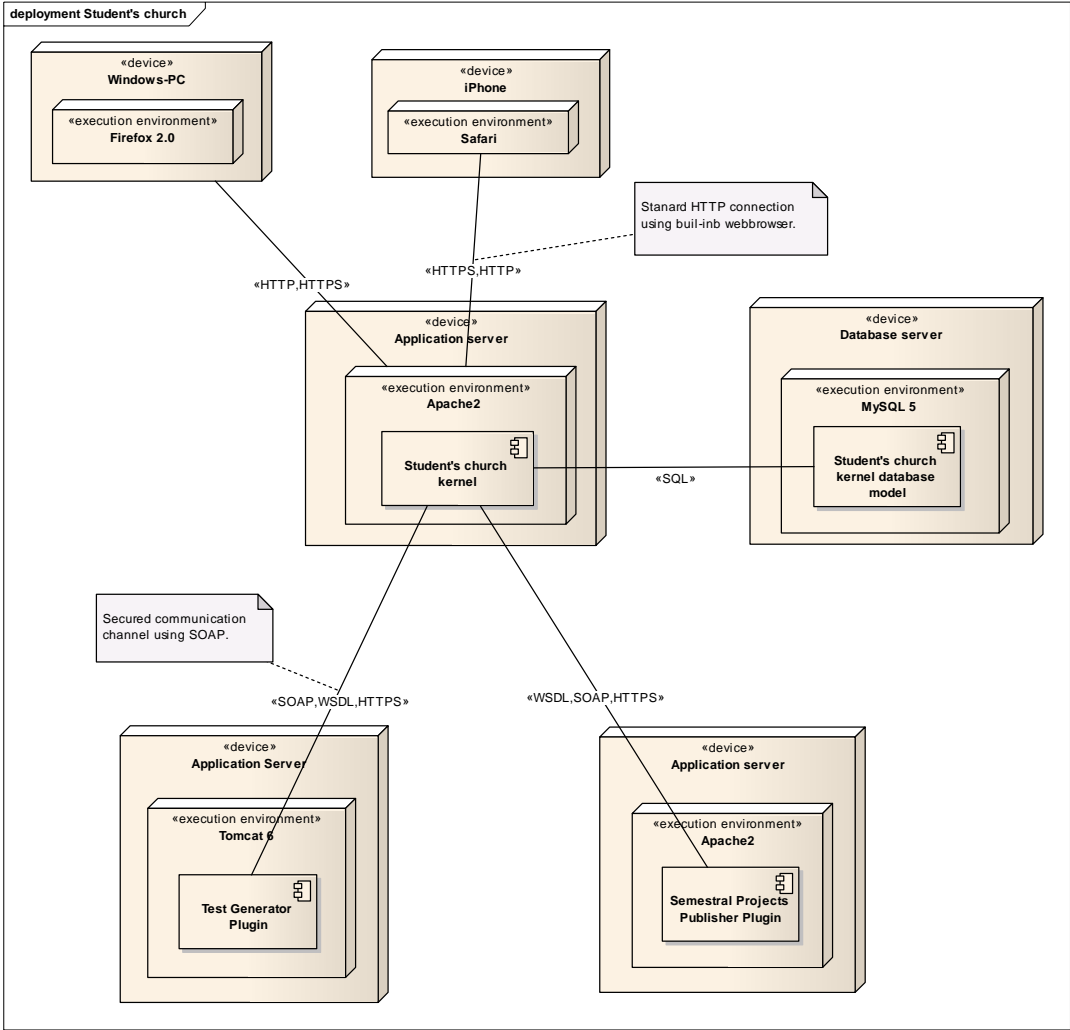


Figure 2: Student's church deployment model

6.1.3 Use cases

Various use cases must have been collected from other members of the development team. Shared functionality was unified and initial use-case model was constructed based on the individual requirements.

Actors

There are four main actors according to requirements of the team.

- Student's church kernel

- represents kernel of application, which could also appear in a role of client
- Plugin
 - Abstract actor used for use cases shared by multiple plugins
- Test evaluator, Test generator, Semestral projects publisher
 - Actors representing particular plugins

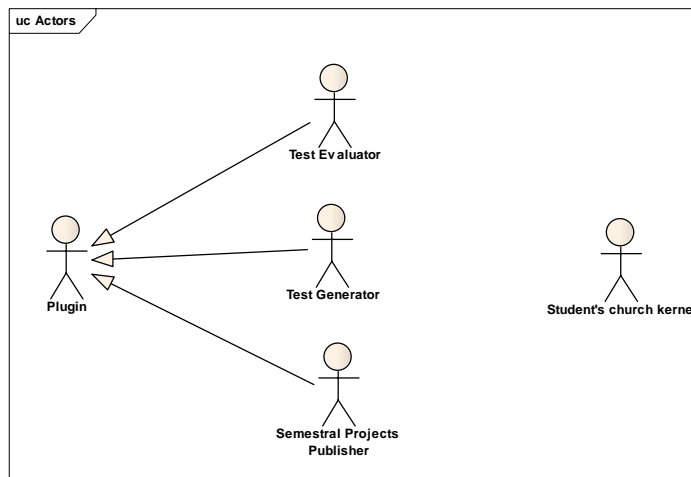


Figure 3: Actors

General use cases and scenarios

General use cases have one abstract actor called plugin and covers majority of common functionality needed by multiple plugins.

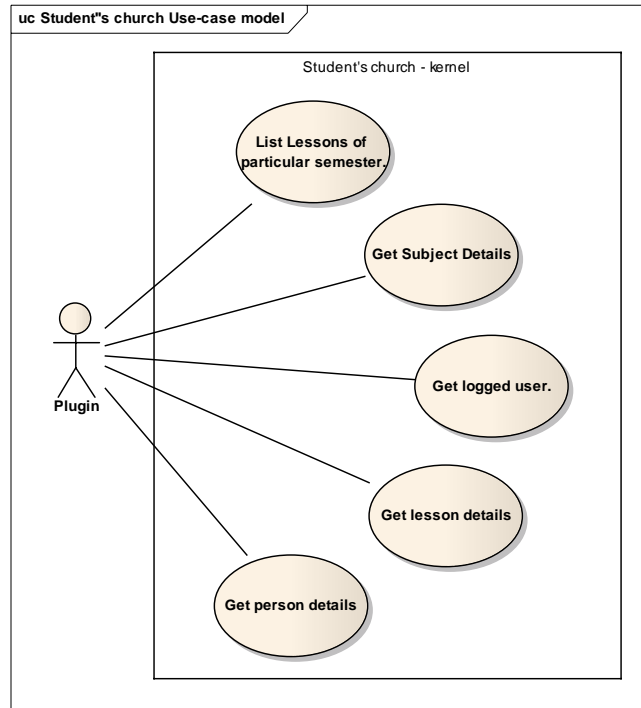


Figure 4: General use-cases

List lessons of particular semester

1. Plugin calls service with parameter semesterID
2. Service returns an array of lessons
 - a. Lesson is a complex data type and has specific parameters

Get subject details

1. Plugin calls service with parameter subjectID
2. Service returns a subject
 - a. The subject is again a specific complex type

Get logged user

1. Plugin calls service without any parameter
2. Service returns a complex type matching login session which is currently active in Student's church kernel

Get lesson details

1. Plugin calls service with parameter lessonID
2. Service returns a complex type matching lesson with given ID

Get person details

1. Plugin calls service with parameter personID
2. Service returns a complex type matching person with given ID

Plugin specific use cases and scenarios

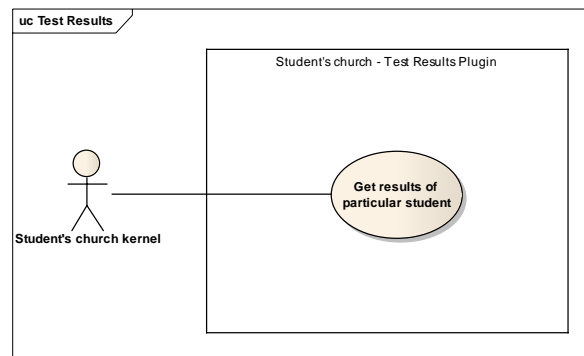


Figure 5: Test results plugin use case

Get results of particular student

1. Student's church kernel call service with parameters studentID and testID
2. Service returns a result of test (in points etc...)

6.2 Mantichora

Mantichora is the second project this work was supposed to solve communication infrastructure of. Whereas Student's church team met on regular meetings, cooperation with Mantichora was more complicated and first meeting of members took place only one month before bachelor theses deadline.

It was not possible to finish the implementation because of amount of activities connected to Student's church part and time requirements on team meetings. This work brings only analytical background for the model of communication of Mantichora.

6.2.1 Requirements

File storage

File storage should enable all the system parts to simply share any data (3D scene source, configuration XML ...).

Negotiation of settings

This part of system should bring possibility of automated negotiation of important parameters e.g. communication speed, port number.

6.2.2 Deployment model

File storage

Deployment model of the file storage is quite simple. The same model represents currently used enterprise applications like Amazon Simple Storage®.

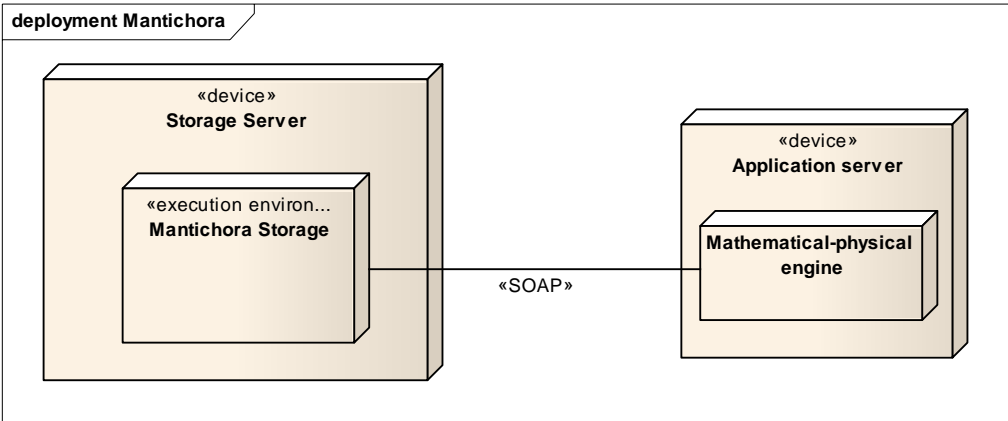


Figure 6: Data Storage Deployment

Negotiation module

Negotiation model brings new features to whole analysis. Firstly parameters of custom communication protocol are transferred using Negotiation module and secondly the canal is created.

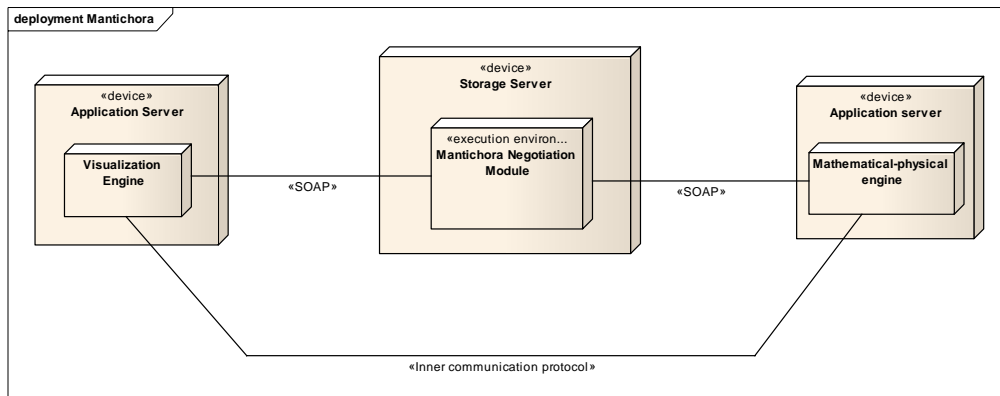


Figure 7: Negotiation module deployment

6.2.3 Use-cases

Data storage

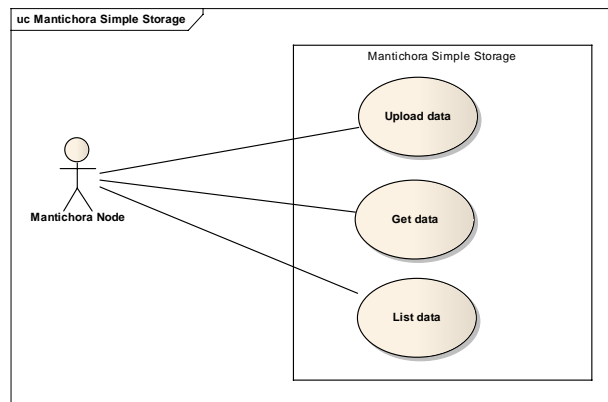


Figure 8: Data storage use cases

Upload Data scenario

1. User selects file to upload
2. Application create the File object (name, size, file type) and sends the File object to data storage
3. Data storage saves the file and sends confirmation

Get Data

1. User selects file to download
2. Data storage reads the file, create the File object (name, size, file type) and sends the File object to the user

List Data

1. User calls method list data with a parameter representing actual path
2. Data storage reads the specified path
3. Data storage returns all directories and files in specified path

Configuration negotiation

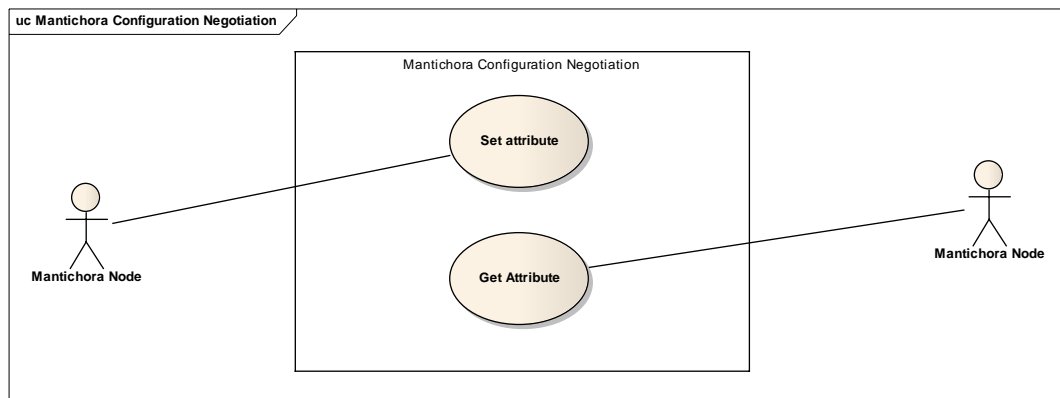


Figure 9: Content negotiation use cases

Set attribute

1. User calls set attribute function (two parameters – name and value)
2. Negotiation engine stores the attribute and sends confirmation

Get attribute

1. User calls get attribute function (one parameter - name)
2. Negotiation engine sends value of attribute of given name

7. Implementation

The next step in the project was the implementation of analyzed use-cases.

7.1 Class Model

The original class model and furthermore also database model of Student's church kernel are described in detail in Hunka (1) and are closely connected to former implementation of Student's church and also character of data provided by KOS - Component of studies – *“information system supporting the agenda of studies on CTU⁵”* (20). The task was to design appropriate subset of data model used by Student's church matching requirements of each plugin author, security and user access policies and shared functionality requirements.

Following classes were used in the final implementation.

Student's church SOAP Server

This class is used as a SOAP Server functionality representative – its public methods are accessible over SOAP by any other application.

User

Class User represents any user who could appear in Student's church kernel. It's not important of what roles is the user representative of (student, lecturer...) – there are stored only contact and id details.

Results, Result, Exam

Class **Results** was based on requirements of **Test Generator** and **Test Evaluator** plugins. Class uses two other important classes – **Result** and **Exam**. All together classes cover data needed by plugins for creation and evaluation of tests – maximum score, minimum score, score of particular student and other.

⁵ Czech Technical University

Semester, Subject, Lesson

These three classes are used for representation of physical classes taught in particular semester, its organization, attending students and lecturer.

File

This class was required by **Semestral project publisher plugin**. For possibility of transfer of binary files **base64** encoding is used.

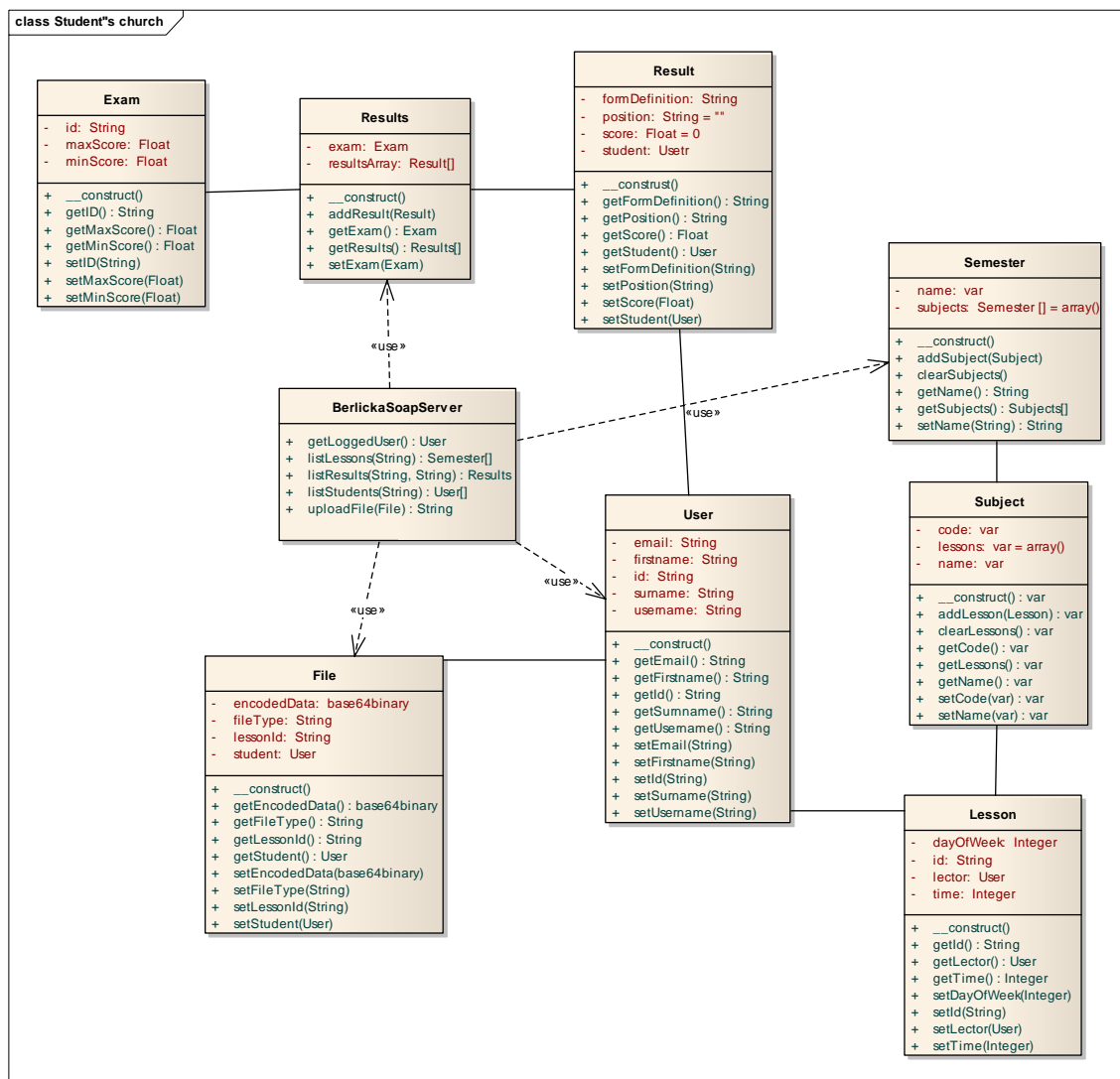


Figure 10: Student's church class model

7.2 Implementation environments

All the implementation environments of particular **web services** were given by team members' requirements. For plugins being currently developed have been used two programming languages – **PHP** and **Java**.

There are many existing implementations which were evaluated for instance by IBM and published on IEEE Web Service Conference 2008 (21). This white paper focused mainly on summary of current situation and available standard open source solutions and their performance.

All three major open-source options – PHP SOAP Extension, AXIS 2 (Java implementation) and AXIS 2 (C implementation) were found viable and competitive. The most common option for implementation of SOAP is according to IBM results PHP although e.g. AXIS 2 provides developers with both full SOAP and RESTFull service more smartly and with less effort. On the other side there are some big enterprise solutions based also on AXIS 2.

7.2.1 PHP

For implementation of PHP part of the project was used standard PHP SOAP Extension. This was caused by following reasons:

- Official implementation by PHP authors
- Standard part of installation packages for both Linux and Windows
- No need for special server configuration
 - In contrary to **Zend Framework (SOAP)**⁶ implementation or **WSO₂ for PHP** (Web Service Framework for PHP)⁷

Main and most important disadvantage of **PHP SOAP Extension** is however the absence of automated generation of **WSDL** and consequently absence of automated negotiation with client applications. This is caused by the architecture of **PHP** itself. **WSDL** uses **XML Schema** for type definition which is very rigid and strict technology for type definition and on the other side there is a weak type

⁶ <http://framework.zend.com/>

⁷ <http://wso2.org/projects/wsf/php>

discipline of **PHP**. As a result of that automated generation of WSDL based only on source code of service is impossible.

There are however existing workarounds of original **PHP SOAP Extension** shortness – providing type definition using source code comments (e.g. special usage of PHPDoc) or manual generation of WSDL.

Finally was chosen an alternative of automatic generation based on PHPDoc comments in source codes by Katy Coe (22). The solution was considered the best because of following reasons:

- Generation of WSDL is automated
- No need for special server-side configuration
- Ease of use

The automated generation of WSDL is crucial because manual creation tends to be faulty and otherwise WSDL needs to be manually modified after each even small change in service or data model. The automated WSDL brings stable and actual description of available services.

Example of usage of special PHPDoc

Proper description of usage is documented in source codes and in tutorials on attached CD. For general knowledge of principles of automated generation of WSDL in PHP based on specialized comments it is good to show short pieces of code.

Service method example

Return type and types of all parameters of specific public function must be covered by *@return* or *@param* notation. All classes needed by function are discovered automatically and could be also forced to be look up by specifying *@internal soaprequires* command.

```

/**
 * Returns logged user (plugin knows who is on the other side)
 *
 * @return User Logged User
 * @internal soaprequires User
 */
public function getLoggedUser() {
    ...
}

```

Class definition example

For processing functions of any class used in SOAP and described by WSDL are the same conditions as for any service method. Both public and private member variables of each class need to be documented (type definition) as well. This is done using *@var Type* notation.

```

class User {
    /** @var string */
    private $firstname;
    ...

    public function __construct() {}

    /**
     * @param string $firstname
     */
    public function setFirstname($firstname) {
        $this->firstname = $firstname;
    }

    /**
     * @return string
     */
    public function getFirstname() {
        return $this->firstname;
    }

    ...
}

```

7.2.2 Java

The whole situation is much simpler in Java implementations according to PHP. As Java is using strict typing, WSDL generation could be done without any restrictions or source code documentation hacks. The IBM report (21) describes features of main open source implementation of web services called Axis 2 by the Apache

(The Apache Software Foundation) (23) and found it complete, viable and competitive implementation.

- *SOAP and RESTFull access*
- *Speed*
- *Low memory foot print*
- *Hot Deployment*
- *Asynchronous Web services*
- *Stability*
- *Component-oriented Deployment*
- *Transport Framework*
- *WSDL support*
- *Add-ons*
- *Composition and Extensibility*

All this features led to usage of Axis 2 as a SOAP implementation for Java.

Test evaluator plugin deployment model (using Axis 2)

This deployment model shows encapsulation of different execution environments – exactly this encapsulation brings easy administration, scalability and stability. Axis 2 takes care of complete SOAP communication and providing WSDL.

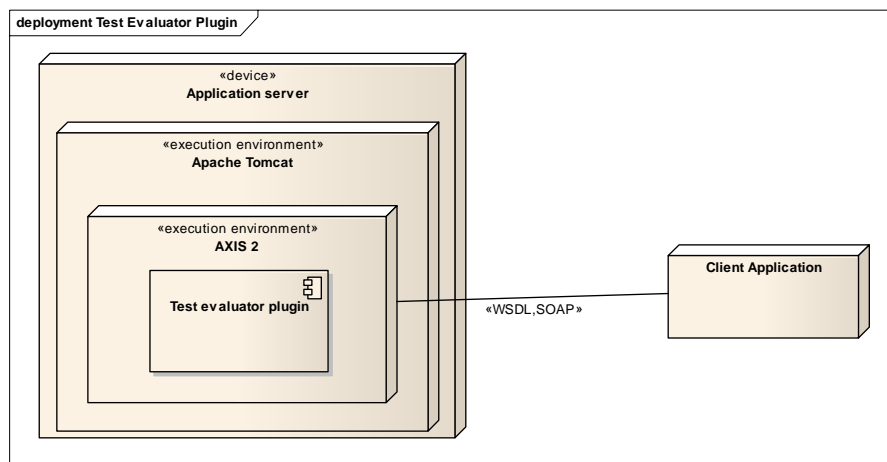


Figure 11: Axis 2 deployment (encapsulation)

7.3 Testing

Whole testing had to be divided into two separate parts: **unit testing** (business logic testing, interface testing) and **architecture & SOAP design testing**. This was necessary according to different approaches which had to be put in practice. Both parts also involved different team members.

7.3.1 Unit testing

Unit testing of specific business logic of Student's church kernel or plugin is **fundamentally out of bounds of this work**. Tests must be targeted (in Java, PHP or any other technology) to specific business logic of given method or whole given class. This work analyzes the architecture of communication, but not the business logic and code of separate parts. Consequently, this part of testing must be covered by authors of separate parts.

7.3.2 Architecture & SOAP design testing

In contrary to unit testing the issue of architectural testing and testing of the design of SOAP implementation must be covered directly by this work. However there is no such an obvious tool as unit testing for validation of given model and services (in fact correctness and completeness of **WSDL**). There are however standard approaches. Mathematical validation (via proof) is not possible – there is not such a precise setting. Software engineering methodologies offer so called informal and formal code reviews. That is the way the proposed model and its implementation were validated and tested in this case.

Formal code review was considered inappropriate because of its time and formal output requirements. On the other side team meetings, which took place ordinarily once a week, were found to be a good opportunity for informal code reviews. Each week was focused on new model parts based on increasing requirements by other team members – **accuracy and completeness** were discussed. Publication of new source codes and UML⁸ diagrams on a team wiki were preceding each meeting.

⁸ Unified Modeling Language

Informal review

The reviews were an ordinary part of each meeting. Progress from last time was discussed. The most stressed were class model and range of offered services. Other discussion included implementation environment related problems and implementation itself.

7.3.3 Results of informal reviews

Made informal reviews produced various results and brought more stable and complete integration of web services (SOAP, WSDL).

Model based requirements

- Requirements on additional classes and additional parameters of existing classes
- Unification of identifiers and its types (entityID)
- Adapter design pattern (link between semesters and taught courses)

Implementation bugs and optimizations

- PHP cache problems (out-of-date WSDL)
- Differences between XML serialization in PHP and Java (XML serialized by PHP could contain invalid characters which resulted in exceptions when parsing in Java)
- Bad configuration of WSDL provided by PHP SOAP Server – invalid complex types as return values in WSDL
- Invalid structure of special comments in PHP resulting in an incomplete WSDL

All this features were identified particularly during informal code reviews and solved.

8. Security

The design of security policies in Student's church is another important part of this work. Protection of personal data, user access policy, user identification and robustness against injecting of fake data (results of tests etc...) are a priority.

Model of deployment of Student's church brought a quite good view of communication channels and its security issues. The whole system would have multiple online user access interfaces (mobile, web browser) and also SOAP access interfaces (on side of kernel, plugins). The principles connected to online communication which must be considered are *invisible security barrier and cross site scripting* (24).

Other security issues shared by almost any software project are common and described in many other studies, tutorials or books and are not necessary to be discussed or mentioned.

8.1 Data protection

Multiple standards cover encryption of communication between system components – kernel, plugins and also clients. One big advantage of proposed and also realized architecture is that communication “**server – plugin**” and also “**server – client**” uses HTTP protocol. Both cases could be therefore solved at once.

Standard way of securing HTTP communication is usage of HTTPS. *HTTPS is a combination of HTTP and cryptography protocol* (25). There are both advantages and vulnerabilities but practical usage and technological support is so wide that it would be impractical to use primarily other technology.

HTTPS could be easily used in **Student' church project** - HTTPS is available in almost all web browsers and in many networking libraries. Configuration of web servers or software components is prepared and well documented.

All the communication between server and plugins, server and clients and clients and plugins should be encrypted using HTTPS.

8.2 Invisible security barrier

Invisible security barrier is a known security principle connected to networking. All the data that leaves solid structure of any application must be validated – even repeatedly. Often vulnerability is a situation where the data are validated for the first time but after that also breaks the barrier and secondly are not validated again.

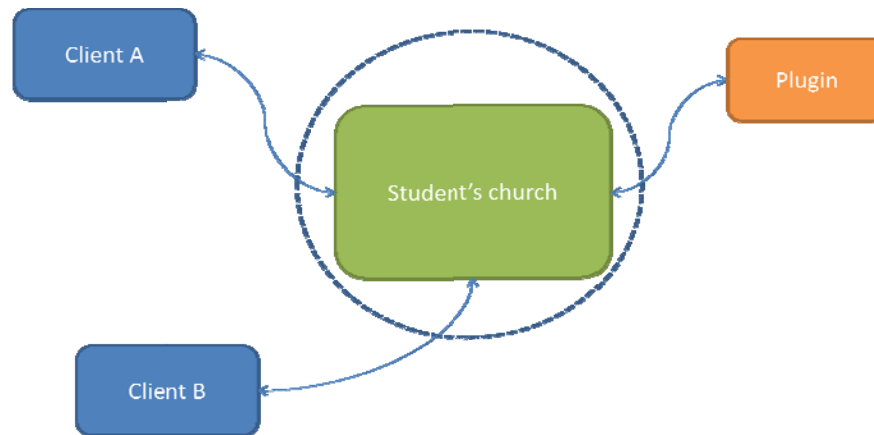


Figure 12: Invisible security barrier schema

There are two situations in the Student's church project, which are different: **Client – Kernel** communication and **Plugin – Kernel** communication.

8.2.1 Client - Kernel

HTTP itself is stateless protocol. Some kind of authentication must be done at some kind at the beginning of each call. Client-kernel communication in Student's church is authenticated by username and password and then by combination of cookie and session. All the authentication must be encrypted (HTTPS).

8.2.2 Plugin – Kernel

This case could be solved similar way to previous one. Plugin could have some user id and password and authenticate the same way as client. But more standard solution is authentication based on security certificates. All the requests must be digitally signed and authenticity of both server and plugins is granted.

Invisible security barrier must be considered when programming both kernel and plugins. All the data transferred through the security barrier must be validated every time. Client-kernel communication must be secured using HTTP and authentication is accomplished by initial login and sessions.

Plugin-kernel communication must be secured using HTTPS. Authentication should be done using signed requests by private key of kernel or plugin. Certificate authority must be chosen carefully.

8.3 Cross site scripting

This security feature is important for business logic of applications not the model of whole information system. All user input must be validated and all data should be controlled also according to their sense – majority of input data should not contains any special characters like >,” or ;. Also input data should not ordinary contain characters outside printable characters (like byte 000 – end of string in C and C++) or any other special characters. **Functional input filtering is a key for cross site scripting robustness. White-list filter policy should be put in practice.**

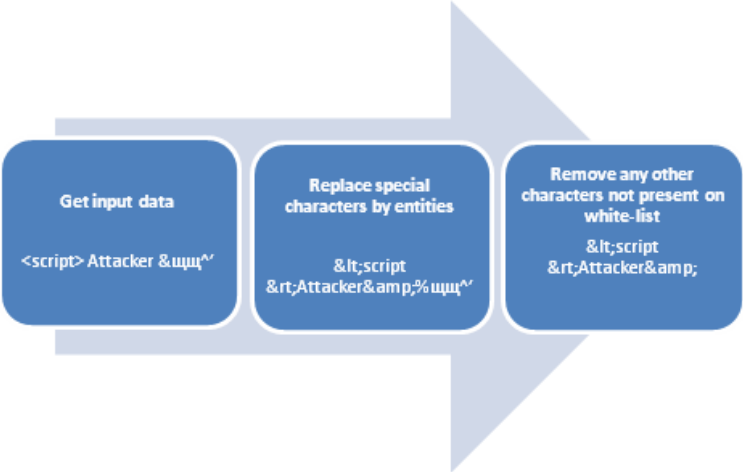


Figure 13: Input filter process flow

8.4 Web services and security features

Security was also solved by authors of web services. WS-Security protocol was developed originally by IBM, Microsoft and VeriSign and solves end-to-end security issues (26). WS-Security provides wider range of security features than simple use of HTTPS or any other mean of TLS⁹.

⁹ Transport Layer Security

9. Conclusion

This work analyzed various existing technologies to provide Student's church and Mantichora projects with a plug-in support and also a possibility of system integration. The most appropriate architecture was found to be SOA¹⁰ because of mentioned reasons (see above) in comparison to various other solutions.

The best existing implementation of SOA in terms of usage in Student's church and partially also Mantichora is SOAP and consequently web services – mainly it is an industry standard with a wide support and a good documentation.

Analysis brought detailed view of the design of the model of communication. This model was reviewed several times and finally implemented for Student's church project. Majority of model were programmed in PHP as well as the Student's church kernel and parts were implemented also in Java (client to Student's church kernel functionality, special server-side functions of a plugin).

A number of informal reviews helped to create a stable and complete solution. Several minor and also severe bugs or model weak places were found and solved. Particular unit tests are the issue of each single developer or pair of developers - business logic of separate applications was not and could not be the task of this bachelor thesis.

Documentation of implemented features is given to developers on both attached CD and project website (<http://berlicka.benefio.cz/wiki>).

9.1 Future progress

This chapter should bring view of possible ways for further development of both projects (knowing the background) as well as recapitulate weak places in each project with a proposal of possible improvement.

¹⁰ Service Oriented Architecture

9.1.1 Project management

The most important thing to be improved in the near future is in my humble opinion project management of both projects. Regular meetings are a good start but are for sure not enough in this kind of project (5-6 cooperating developers, different working time and working environment). At the beginning I focused on the project management as well but I was not enough successful.

I tried to introduce basic team tools to the project but only one appeared to be used regularly.

Team Wikipedia

Team Wikipedia (<http://berlicka.benefio.cz/wiki>) was the only application practically used by team members. Team members used Wiki to comment common parts of their applications, also to distribute shared parts of source codes and to publish short manuals of usage.

Bug and task tracking software

Bug and task tracking software (<http://berlicka.benefio.cz/mantis>) is important for easier communication between developers and also between developer and supervisor. I used system **Mantis** connected also to source code repository. I was not successful in introducing this to other developers – almost no tasks were given using this system and the same situation was for bug reporting. This situation was also affecting the communication delays which brought some problems in the second half of the project development.

Source code management software

Source code management software (like **Subversion**) is also an important collaboration tool for developers. It allows developer to automatically backup his/her software, to store history of the project (all changes done to source codes) and as well to distribute latest source codes to partners in real time. Available SVN I provided was also not used. This is but the last important point because majority of source codes were because of high independency of plugins not needed by other team members.

Project Management summary

There should be introduced a position (for instance another Bachelor Thesis) of Project manager, who will be responsible for proper functionality of collaboration tools, coordination of activities of single developers, planning of meetings and informal code reviews (or either formal code reviews if it were appropriate). This person could provide team with short introduction to each shared tool and then force developers to use it regularly – to enable smoother, faster and more predictable development praxis.

9.1.2 Student's church

Student's church project has already shown its usability in praxis. The future development is currently easier particularly because of implementation of **SOA**¹¹ described in this work.

Next big task is to create a **user interface transfer** – possibility of transferring plugin user interface directly to Student's church kernel environment. This task had been thought to be done in current phase but later was decided to postpone the implementation to the near future. It could also be done via web services and I hardly recommend that – particularly because of security reasons and principles mentioned in security analysis chapter.

9.1.3 Mantichora

Mantichora is the most ambitious project of all mentioned. Current situation as well as future plans could be well described by simple schema.

¹¹ Service Oriented Architecture

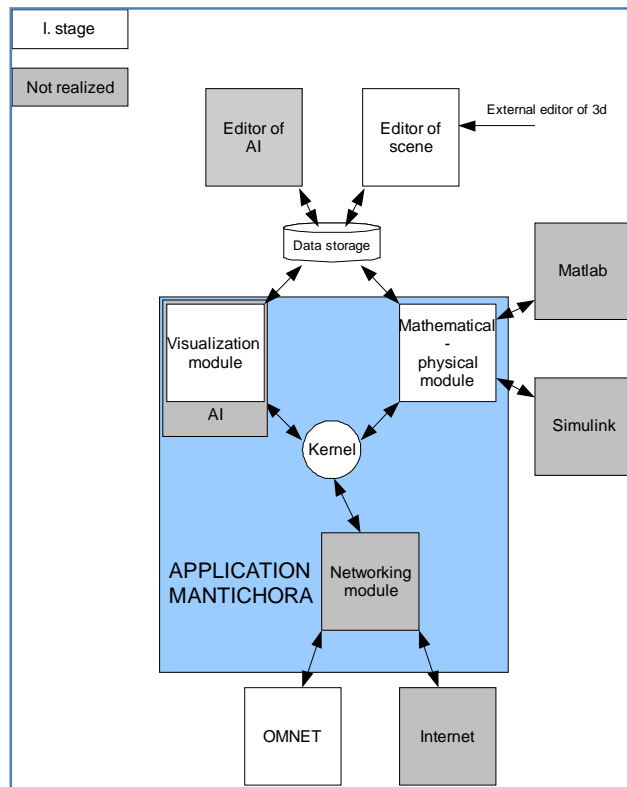


Figure 14: Mantichora development schema, by Ing. Jiří Chludil

The networking and communication issues which will need to be solved with a special attention will be especially real time communication channels. These channels will in fact realize connection between mathematical-physical module and visualization module. This kind of communication has special requirements different from ordinary requirements connected to information system integration.

10. Bibliography

1. **Hunka, Jiří.** *Studentova Berlička, Bakalářská práce.* Praha : FEL ČVUT, 2007.
2. Monolithic application. *Wikipedia.org.* [Online] 5 24, 2009.
http://en.wikipedia.org/wiki/Monolithic_application.
3. Software componentry. *Wikipedia.org.* [Online] 5 24, 2009.
http://en.wikipedia.org/wiki/Software_componentry.
4. Three-tier model. *Wikipedia.org.* [Online] 5 24, 2009.
[http://en.wikipedia.org/wiki/Three-tier_\(computing\)](http://en.wikipedia.org/wiki/Three-tier_(computing)).
5. Plugin. *Wikipedia.org.* [Online] 5 24, 2009. <http://en.wikipedia.org/wiki/Plugin>.
6. Peer to peer. *wikipedia.org.* [Online] 5 24, 2009.
<http://en.wikipedia.org/wiki/Peer-to-peer>.
7. Service Oriented Architecture. *Wikipedia.org.* [Online] 5 24, 2009.
http://en.wikipedia.org/wiki/Service_Oriented_Architecture.
8. Loose coupling. *Wikipedia.org.* [Online] 5 24, 2009.
http://en.wikipedia.org/wiki/Loose_coupling.
9. SOAP. *Wikipedia.org.* [Online] 5 27, 2009.
[http://en.wikipedia.org/wiki/SOAP_\(protocol\)](http://en.wikipedia.org/wiki/SOAP_(protocol)).
10. Remote Procedure Call. *Wikipedia.org.* [Online] 5 27, 2009.
http://en.wikipedia.org/wiki/Remote_procedure_call.
11. Distributed Component Object Model. *Wikipedia.org.* [Online] 5 27, 2009.
http://en.wikipedia.org/wiki/Distributed_Component_Object_Model.
12. Common Object Request Broker Architecture. *Wikipedia.org.* [Online] 5 27, 2009. <http://en.wikipedia.org/wiki/CORBA>.
13. SOAP Introduction. *W3Schools.com.* [Online] 5 27, 2009.
http://www.w3schools.com/soap/soap_intro.asp.

14. **Fielding, Roy Thomas.** Architectural Styles and the Design of Network-based Software Architectures. [Online] 2000.
<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.
15. Web Services. *Wikipedia.org*. [Online] 5 29, 2009.
http://en.wikipedia.org/wiki/Web_service.
16. Web Services Introduction. *W3schools.com*. [Online] 5 29, 2009.
http://www.w3schools.com/webservices/ws_intro.asp.
17. **Nilo, Mitre and Ericsson, Lafon Yves.** SOAP 1.2 Primer. *W3.org*. [Online] 4 27, 2007. <http://www.w3.org/TR/2007/REC-soap12-part0-20070427/>.
18. **Booth, David and Liu, Canyang Kevin.** WSDL Primer. *w3.org*. [Online] 6 26, 2007. <http://www.w3.org/TR/wsdl20-primer/>.
19. UDDI Specifications. *oasis-open.org*. [Online] 5 29, 2009. <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm>.
20. Komponenta Studium. *CVUT.cz*. [Online] 6 2, 2009.
<http://www.cvut.cz/informace-pro-zamestnance/is/kos>.
21. **Toyotaro Suzumura, Scott Trent, et al.** Performance Comparison of Web Service Engines in PHP, Java, and C. *IEEE International Conference on Web Services*. [Online] 2008.
http://www.trl.ibm.com/people/mich/pub/200809_icws2008wsperf.pdf.
22. **Coe, Katy.** Automatic WSDL Generation in PHP 5. [Online] 7 27, 2006.
<http://www.djkaty.com/php-wsdl>.
23. Welcome to Apache Axis2/Java. [Online] [Cited: 6 6, 2009.]
<http://ws.apache.org/axis2/>.
24. **Huseby, Sverre H.** *Zranitelný kód*. Brno : Computer Press, a.s., 2006.
25. HTTP Secure. *Wikipedia.org*. [Online] [Cited: 6 6, 2009.]
<http://en.wikipedia.org/wiki/Https>.
26. WS-Security. *Wikipedia.org*. [Online] [Cited: 6 6, 2009.]
<http://en.wikipedia.org/wiki/WS-Security>.

27. Zákon č. 101/2000 Sb., o ochraně osobních údajů (účinné znění). *Úřad pro ochranu osobních údajů*. [Online] 4 4, 2000.
<http://www.uoou.cz/uoou.aspx?menu=4&submenu=5&loc=20>.
28. **Smith, Jerry**. 10 Measures for Successful SOA Implementations . *SOAWorld Magazine*. [Online] 8 5, 2008. <http://soa.sys-con.com/node/631831?page=0,1>.
29. **Haas, Hugo**. Web Services Glossary. *W3.org*. [Online] 2 11, 2004.
<http://www.w3.org/TR/ws-gloss/>.
30. Representational State Transfer (REST). *Wikipedia.org*. [Online] 29 5, 2009.
http://en.wikipedia.org/wiki/Representational_State_Transfer.
31. Webservice schema. *Wikipedia.org*. [Online] 6 6, 2009.
http://en.wikipedia.org/wiki/File:Webservice_xrpc.png.

Appendix A

Attached CD

There are various sections on an attached CD and all the content is described in detail in "**readme.txt**". The CD contains all the source codes and also whole **Netbeans** project packages. Secondly the CD contains manuals for both PHP and Java introducing both server and client side of the SOAP and generation of WSDL; simply follow an appropriate link in the file "**index.html**". Last thing is the electronic version of this bachelor thesis in PDF named "**Marek-Sacha-Bachelor-Thesis-2009.pdf**" and in DOCX named "**Marek-Sacha-Bachelor-Thesis-2009.docx**".