Insert here your thesis' task.

Czech Technical University in Prague

Faculty of Information Technology

Department of theoretical computer science

Master's thesis

Admission procedure Automatic processing of applications for master's study program

Bc. Ján Ondrušek

Supervisor: Ing. Tomáš Kadlec

11th June 2012

Acknowledgements

I would like to thank my family and friends for support during writing this thesis.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act 60 no. 121/2000 (copyright law), and with the rights connected with the copyright act included the changes in the act.

In Prague 11th June 2012 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Ján Ondrušek. *Admission procedure Automatic processing of applications for master's study program: Master's thesis.* Czech Republic: Czech Technical University in Prague, Faculty of Information Technology, 2012.

Abstract

Primary aim of this thesis is to analyse Conditions for admission and Dean's directive for admission process to master's study programme at CTU FIT. Implement RESTful API, which exposes backend functionality for admission processing using Business Process Management.

Keywords Admission procedure, RESTful API, BPM, jBPM, Spring, Spring Roo

Abstrakt

Primárnym cieľom tejto diplomovej práce je analyzovať Řád přijímacího řízení ČVUT a Směrnici děkana pro přijímací řízení na ČVUT Fakultě informačních technologií. Implementovať RESTful API, ktoré vystaví funkcionalitu backendu pre prijímací proces s použitím Business Process Management stroja.

Klíčová slova Spracovanie prihlášok, RESTful API, BPM, jBPM, Spring, Spring Roo

Contents

List of Figures

List of Tables

Prologue

Motivation and objectives

Every year, hundreds of high school graduates apply for studies at Czech Technical University, Faculty of Informatics. This raises certain requirements, including managing, storing, analysing and processing of all these applications. Each application has its own life cycle, which begins with filling out an on-line form and continues through various steps which an applicant has to pass. The life cycle ends when a decision of acceptance is delivered to the applicant and he either enrolls in the studies or not.

We live in the world of new era of the Internet. Everything goes on-line, web and the latest trend - everything goes mobile. People want things to happen very quickly. They want to access all the information fast, now.

Students and applicants are no different. They expect from this prestigious University, especially from Faculty of Informatics, the most modern and useful gadgets when it comes to software and web.

How do things work now

Currently all applications are processed rather manually. Many man days of administrative work are consumed during the process. Although an electronic form is filled in and submitted by an applicant, the rest of actions almost exclusively fall into the hands of Study Department staff. Some of the work is handled by simple scripts or other utilities. The question is: Why don't we do most of the work automatically?

This work is monotonous and can even lead to men's frustration.

What should be achieved - the goals

Courses at Faculty of Informatics teach its students to handle various programming languages, web technologies and techniques. We all know what to expect from a working web application and good looking one is a bonus. This is why knowledge of faculty's students should be used for good of their suc-

cessors. Fast, reliable, informative and functional system will make them feel more comfortable and perhaps could even save some precious time.

Ideal state would be to accept on-line applications and automatically generate invitations for applicants, that should attend a test. After the test, process all results and generate a decision of acceptance letter for all who passed the test or are accepted without it. The only manual interventions that will remain is to accept apology, appeal and insert the letters into the envelopes.

Pragmatically goals of this thesis could be summarized as follows:

- familiarise with RESTful best practices, patterns and anti-patterns
- familiarise with Business process management (BPM) with main focus on jBPM
- implement RESTful Application Programming Interface (API) (back-end) according to functionality requested by Android and Web UI teams
- implement admission processing using Java and jBPM processing machine
- explore new and modern Java (JEE) technologies
- follow modern development methodologies
- perform tests during and after development
- use exclusively Open Source software and tools

Let's make things better

Taking the above written into account, this might be a good idea for a master's or bachelor's thesis. However if we want to use all available technologies that have become popular in past years and automatize the majority of admission processing, it turns out to be a very complex project. So why not to create several teams and split necessary work into multiple, both bachelor's and master's, thesis?

This is how project Přiříz was born. It includes web interface for both students and Study Department staff, native Android application and RESTful API with BPM processing machine, which is the subject of my master's thesis.

Structure of this work

Basically, I could divide my work into these main parts, which are then further split into chapters:

- Theoretical introduction is covered by chapters 1 and 2. Following parts will largely draw on the information contained here.
- Analytical part talks about architecture of the whole ecosystem with main focus on RESTful API. Describes technologies used, methodologies applied and tools commonly used during development and testing phases.
- Implementation and unit testing
- Integration and regression testing
- Results and conclusion

Appendices at the end of the document are referred directly from the text within the chapters. Smaller figures, tables or other objects are put directly into the content.

CHAPTER 1

RESTful API with JAX-RS

Nowadays, Internet consumers demand fast growth of various services and integration of their favourite ones. As en example I can point out synchronization of contact list between very popular social networks, e-mail providers and phone contact lists.

Other example may be growing amount of `mashups`[1] and uncountable number of `startups`[2], who often provide RESTful or different type of public API.

1.1 Talking about REST, what is it?

REpresentational State Transfer (REST) or RESTful programming is not defined by any official standard and there are no official guidelines or rules for it. So what is it then? It is and architectural and programming style for Web, where a set of constraints is defined. Lots of text has been written about it during past years and describing the whole idea of REST is out of scope of this master's thesis. I can however try to point out the most significant, important and basically, what I personally managed to adopt.

1.1.1 Main principles of REST, RESTful web service

There are several architectural principles that one should keep in mind when thinking of REST [1, p. 3]:

[1] Applications that are created via combination of multiple different services. Such application, almost exclusively web based, can be created very quickly by consuming several APIs. Not necessarily from the same provider.

[2] Constantly rising amount of web applications that focus on fast growth of attracted users. They offer various services, which are often very innovative and experimental. One successful example is popular social network and my favorite information channel - Twitter.

- **Addressable resources** The key abstraction of information and data in REST is a resource, and each resource must be addressable via a Uniform Resource Identifier (URI).
- **A uniform, constrained interface** Use a small set of well-defined methods to manipulate your resources.
- **Representation-oriented** You interact with services using representations of that service. A resource referenced by one URI can have different formats. Different platforms need different formats. For example, browsers need HTML, JavaScript needs JSON (JavaScript Object Notation), and a Java application may need XML.
- **Communicate statelessly** Stateless applications are easier to scale.
- **Hypermedia As The Engine Of Application State (HATEOAS)** Let your data formats drive state transitions in your applications.

HATEOAS is often understood as a core principle of REST. It carries an idea of resource representation via links and stateless implementation of services.

RESTful web services are the result of applying these constraints to services that utilize web standards such as URIs, Hypertext Transfer Protocol (HTTP), Extensible Markup Language (XML), and JavaScript Object Notation (JSON).

1.1.2 Back to the roots, HTTP is reborn

Service-oriented architecture (SOA) has been in this world for a long time. Many different approaches and technologies exist to implement it. From those worth to mention: DCE, CORBA, Java RMI, ... They offer robust standards, one can build large, complex and scalable systems on top of it, but there is a cost. They often bring huge complexity and maintenance requirements into place.

Currently, when one says SOA, it often evokes Simple Object Access Protocol (SOAP) in a mind that spent several years using technologies mentioned above. This however is not a bad thing. SOAP is used very widely and is perfectly suitable for developing services and APIs. But it is definitely not a lightweight technology and it is not ideal for everything. Its most common use case is for server-server communication in enterprise systems.

Nowadays, we need something quickly adoptable, widely spreadable, platform and technology independent and client oriented. This needs a completely different approach and new way of thinking when it comes to SOA. It is about Web, so why not to start with something that is Web, as we see it today, based on? Yes, it is HTTP.

Although REST is not protocol specific, when saying REST it usually automatically means REST + HTTP. No wonder. HTTP is perfectly suitable for client-server SOA, it is just about the way of thinking. It offers transport layer, request-response mechanism, descriptive responses, caching mechanism and many more. It is true that in past years, when various types of web applications started to appear, many web developers limited their thinking and use of HTTP to two basic cases:

- GET a page with URI, perhaps containing a few query parameters
- POST a form

Code 1 HTTP GET request/response example of a standard web page

```
GET /index.html HTTP/1.1
User-Agent: curl/7.24.0 (i686-pc-cygwin) ...
Host: www.google.sk
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: sk-sk,en;q=0.5

HTTP/1.1 200 OK
Date: Thu, 07 Jun 2012 11:25:15 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-2
Set-Cookie: ... expires= ...; path=/; domain=.google.sk
Set-Cookie: ... expires= ...; path=/; domain=.google.sk; HttpOnly
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Transfer-Encoding: chunked

<!doctype html><html ...><head> ... <body> ...
```

The example above 1 shows most common HTTP request and response, when browsing the web via standard web browser. It requests object **/index.html** using **GET** method placed on host **www.google.sk**. My client also put several HTTP headers into the request:

- **Accept:** text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
- **Accept-Language:** sk-sk,en;q=0.5

Also the request does not contain any request body, as it is GETting information from the server.

The response of the message received is 200, which means OK - success. An overview of all available HTTP response codes can be found on-line at [8]. **Content-type** header of the response message says that the body received is of type HTML.

RESTful web service needs more than that and luckily HTTP offers much more. It will be better to point out its features in a relationship to each of REST's architectural principles.

1.1.3 Addressable resources, URIs and links

Each **resource** in a system should be reachable through a **unique identifier**. When reflected to the idea of REST and HTTP, URIs will automatically come to mind. The format of a URI looks like this [11]:

```
<scheme>://<authority><path>?<query>
```

The above is a citation directly from RFC, but for purposes of this master's thesis should be rewritten into more detailed form:

```
<scheme>://<host>:<port>/<path>?<queryString>#<fragment>
```

Where in the RESTful world these parts usually mean:

- **scheme** typically **http** or **https**
- **host** aka server name, e.g. **fit.cvut.cz** and **port**
- **path** to the resource on server, e.g. **/admission/123-456-01**
- **queryString** after the **?** is typically used for a set of resources and can be a page number, number of items in the set, or a filter definition and many more, e.g. **?page=1&limit=10**
- **fragment** after the # usually points to a certain place in a document

An example of such URI pointing to a set of resources may be:

```
http://pririz.is.fit.cvut.cz:9090/admission/services/admission
?page=2&count=20
```

An example of URI pointing to a concrete single resource:

```
http://pririz.is.fit.cvut.cz:9090/admission/services/admission
/123-456-01
```

Characters allowed in URI string are all alphanumeric, comma, dash, asterisk and underscore. Space is converted into plus and other characters are encoded using specific schema into a two digit hexadecimal number, which is appended to the % character.

1.1.4 The uniform, constrained interface, HTTP methods

This is probably the prettiest part of a relationship between REST and HTTP. It may be a bit difficult to adopt this principle for a person, who spent a couple of years developing CORBA or SOAP services. There is a finite set of HTTP methods and all REST operations have to stick to it. All other parameters describing operations must be omitted from URI.

Let's see what HTTP offers:

Method	`Idempotent`[a]	`Safe`[b]	Operation(s)
GET	yes	yes	read - query information from a server
POST	no	no	write, update - both can change a server state in a unique way each time executed
PUT	yes	no	update for the known resource, updating the same resource more than once does not effect it
DELETE	yes	no	delete - removes resource
HEAD	yes	yes	read without response body, returns only response headers
OPTIONS	yes	yes	information about communication options with server

[a]Idempotent means that no matter how many times you apply the operation, the result is always the same.

[b]Safe means that invoking an operation does not change the state of the server at all. This means that, other than request load, the operation will not affect the server.

Table 1.1: An overview of HTTP methods and their roles in a RESTful service

HTTP contains a few other methods (TRACE, CONNECT), which are unimportant for purposes of RESTful services.

What is more interesting, nowadays a couple of non-HTTP methods have appeared, which may be good for RESTful service design in the future. Namely PATCH (very similar to the PATCH method found in `WebDAV`[3]) and MERGE. According to various sources I have found on the web, namely [5], [6] or [2], they may appear in further HTTP specifications. They are not a part of current HTTP 1.1.

PATCH and MERGE are used for partial update of known resources that contain large amount od data and updating the whole object would be a

[3]WebDAV stands for Web-based Distributed Authoring and Versioning. It is a set of extensions to the HTTP protocol which allows users to collaboratively edit and manage files on remote web servers.

lengthy and ineffective operation. This however can be simulated using POST and specifying detailed path of the resource via URI.

1.1.5 Representation-oriented

I already described that each resource has its own URI and client-server principle using HTTP. Its methods allow the client to receive current representation via GET method, remove it from server via DELETE or change the representation via POST and PUT methods. Concrete representation can be received in JSON, XML, YAML or any other format one can imagine.

Representation format is agreed between client and server in a RESTful system interaction. HTTP offers such feature by specifying **Content-Type** header. Its value string is represented by Multipurpose Internet Mail Extensions (MIME) format:

```
<type>/<subtype>[;name=value;name=value...]
```

An example may be:

```
text/html; charset=utf-8
text/xml
text/json
application/xml
application/json
```

To choose preferred format(s), client can specify **Accept** HTTP header in a request. Now it becomes more clear how REST and HTTP perfectly fit each other. Together they offer addresability, method choice and object representation format.

1.1.6 Communicate statelessly, (no)sessions

HTTP offers powerful client session management, which is commonly used when browsing the web via traditional web browser. It stores so called **Cookies** when a server asks for it in response headers, which are then sent back to the server with subsequent requests. This is how the server handles stateful interaction with the client over HTTP.

Stateless communication in REST means that there is no client session data stored on a server. In other words, none of the above is performed. It does not mean that RESTful application cannot be stateful, though.

Reason for this is simplicity, which further leads to easily scalable RESTful service. It is generally much less difficult to build a cluster of stateless applications than to handle session replication and possibly another service layer.

1.1.7 HATEOAS

[1, p. 11] Hypermedia is a document-centric approach with the added support for embedding links to other services and information within that document format.

There are several ways how to understand any apply this RESTful architectural principle. One use case is to add hyperlinks when composing complex and large objects. This avoids unexpected server load, delay in response to the client and helps to reference dependent or embedded objects without bloating the response.

An example of server response without any hyperlinks:

```
<terms>
  <term>
    ...
    <registrations>
      <registration>
        <admission>
          <code>96858805</code>
          <type>P</type>
          <accepted>false</accepted>
          ... some huge object containing
          a lot of information
        </admission>
        <term>
          <dateOfTerm>2012-05-10...</dateOfTerm>
          <room>BS</room>
          <capacity>1500</capacity>
          <registerFrom>2012-05-03...</registerFrom>
          <registerTo>2012-05-08...</registerTo>
          <apologyTo>2012-05-08...</apologyTo>
          ... another huge object containing
          a lot of information, graph can be even circular
        </term>
      </registration>
      ...
    </registrations>
    ...
  </term>
  ...
</terms>
```

Let's apply embedded hyperlinks on the document above:

```
<terms>
  <term>
```

```
    ...
    <registrations>
      <registration>
        <admission>
          <link href="http://.../admission/96858805"
          method="GET" rel="admission" />
        </admission>
        <term>
          <link href="http://.../term
          /dateOfTerm:2012-05-10T14:22:00/room:BS"
            method="GET" rel="term" />
        </term>
      </registration>
      ...
    </registrations>
    ...
  </term>
  ...
</terms>
```

This concept of HATEOAS is called aggregation. But it isn't everything. In a case that the server would return thousands of term objects and each of them would contain thousands of registrations including admissions, the response would be, again, very large. However, there is one even more interesting part of HATEOAS - the „engine".

Core of the engine principle is not to return the whole set of object available, but just a subset of it and to tell the client, where to find the rest:

```
<terms>
  <count>5</count>
  <totalCount>123</totalCount>
  <link href="http://.../term?page=3&count=5" method="GET"
    rel="next" />
  <link href="http://.../term?page=1&count=5" method="GET"
    rel="previous" />
  <term>
    ...
    <registrations>
      <registration>
        <admission>
          <accepted>false</accepted>
          <link href="http://.../admission/96858805"
            method="GET" rel="admission" />
        </admission>
        <term>
```

```
          <link href="http://.../term
          /dateOfTerm:2012-05-10T14:22:00/room:BS"
            method="GET" rel="term" />
        </term>
      </registration>
      ...
    </registrations>
    ...
  </term>
  ...
</terms>
```

This approach saves a lot of server resources and prevents client from unexpected delays due to large responses. Such response should be always returned in constant time, because the request query defines its maximum size - number of objects.

1.2 REST, Java and JAX-RS

[1, p. xiii] The Java API for RESTful Web Services (JAX-RS) is a new API that aims to make development of RESTful web services in Java simple and intuitive. The initial impetus for the API came from the observation that existing Java Web APIs were generally either:

- Very low-level, leaving the developer to do a lot of repetitive and error-prone work such as URI parsing and content negotiation, or
- Rather high-level and proscriptive, making it easy to build services that conform to a particular pattern but lacking the necessary flexibility to tackle more general problems.

JAX-RS is one of the latest generations of Java APIs that make use of Java annotations to reduce the need for standard base classes, implementing required interfaces, and out-of-band configuration files. Annotations are used to route client requests to matching Java class methods and declaratively map request data to the parameters of those methods. Annotations are also used to provide static metadata to create responses.

JAX-RS also provides more traditional classes and interfaces for dynamic access to request data and for customizing responses.

This is a brief description of what JAX-RS is. Its usage and development approach will be demonstrated in following parts of this master's thesis.

CHAPTER 2

BPM and jBPM

[4] Business process modeling (BPM), sometimes called business process management, refers to the design and execution of business processes.

It does not have to be necessarily used in a context of Information Technology (IT) and software development. Primary field of BPM falls into management, though. Before Information and communication technologies (ICT) was widely spread and automatic software processing was a dream, BPM was manual and paper driven.

But yes, BPM is also closely aligned with the notion of SOA, particularly the emerging W3C web services stack. Whereas the traditional use of a workflow was about the movement of work from person to person within an organization, contemporary BPM processes are built to interact as services with other systems, or even to orchestrate or choreograph other systems, including the business processes of other companies.

2.1 Business Process

A business process is a service, one intended to be called by other systems, and these calls drive its execution. Realizing this fact is one of the first big steps in understanding BPM.

Being algorithmic, a process can potentially be run by some sort of process engine. As long as the process can be expressed in a form that is syntactically and semantically unambiguous that is, in a programming language or other interpretable form the engine can accept it as input, set it in motion, and drive its flow of control. To be precise, the engine creates and runs instances of a given process definition. The steps of the process are called activities or tasks.

2.2 Important process modelling terms

- **Process definition** The basic algorithm or behavior of the process.
- **Process instance** An occurrence of a process for specific input. Each instance of the travel reservation process, for example, is tied to a specific customer's itinerary.
- **Activity or task** A step in a process, such as sending a flight request to the airline.
- **Automated activity or automated task** A step in a process that is performed directly by the execution engine.
- **Manual activity or manual task** A step in a process that is meant to be performed by a human process participant.

The distinction between manual and automated activities is extremely important. At one time, before the reign of software, a business process was completely manual and paper-driven: paper was passed from person to person, and was often misplaced or delayed along the way. Now, much of the process runs on autopilot.

Automated activities generally fall into two categories:

- **Interactions with external systems** e.g., sending a booking request to an airline
- **Arbitrary programmatic logic** e.g., calculating the priority of a manual task

2.3 BPM application

BPM is suited only for applications with an essential sense of state or process that is, applications that are process-oriented. Typical characteristics of a process-oriented application are:

- **Long-running** From start to finish, the process spans hours, days, weeks, months, or more.
- **Persisted state** Because the process is long-lived, its state is persisted to a database so that it outlasts the server hosting it.
- **Bursty, sleeps most of the time** The process spends most of its time asleep, waiting for the next triggering event to occur, at which point it wakes up and performs a flurry of activities.
- **Orchestration of system or human communications** The process is responsible for managing and coordinating the communications of various system or human actors.

2.4 BPM standards

BPM is not that difficult. The business analyst designs the process, the process is run by an engine, and the engine has EAI and human interaction capabilities. Questions appear when it comes to selection of a right solution, design and modelling tools, runtime engine, ...

It is good to look for some well known and widely accepted approach. BPM does not lack standards. The adoption of Business Process Execution Language (BPEL) and Business Process Model and Notation (BPMN) is a recipe for success. According to [4, Ch. 1.3.1], the important BPM standards are:

- Business Process Execution Language for Web Services (BPEL4WS), sometimes shortened to **BPEL**

  A BPEL process is a web service with an associated process definition defined in an XML-based language. The behavior of a BPEL process is to act on, and be acted on by, other processes; put differently, a BPEL process can invoke another web service or be invoked as a web service.

- **Business Process Modeling Language (BPML)**

  From the Business Process Modeling Initiative (BPMI) organization, which is an XML-based process definition language similar to BPEL. BPMN, another specification from BPMI, is a sophisticated graphical notation language for processes. Significantly, the BPMN specification includes a mapping to BPML-rival BPEL, which facilitates the execution of BPMN-designed processes on BPEL engine.

- **Web services choreography**

  Choreography describes, from a global point of view, how web services are arranged in a control view spanning multiple participants. Choreography's global view is contrasted with the local view of process orchestration in languages such as BPEL; a BPEL process is the process of a single participant, and a choreography is the interaction model for a group of participants. Web Services Choreography Description Language (WS-CDL) is the W3C's recommended choreography standard.

- Workflow Management Coalition (WfMC) has published a BPM reference model, as well as a set of interfaces for various parts of the BPM architecture. Though WfMC does not specify a standard graphical process notation, it does provide an exportable XML format called XML Process Definition Language (XPDL); processes built in an XPDL-compliant design tool can run on a WfMC enactment engine.

- **Object Management Group (OMG)** did not aim build a new process language or interface but abstract BPM models conforming to its Model-Driven Architecture (MDA).

- **Business Process Specification System (BPSS)** from OASIS group

  A choreography language, but is built for business-to-business collaborations. In a typical exchange between a buyer and seller, for example, the buyer sends a request to the seller, to which the seller responds immediately with consecutive acknowledgements of receipt and acceptance. When the seller has finished processing the request, it sends an indication of this to the buyer, and the buyer in turn sends an acknowledgement of the indication to the seller.

From the above are only two standards worth considering. BPEL and BPMN, which I am going to describe in analytical part of this work.

2.5 Execution engine - jBPM

[10] jBPM is a flexible BPM Suite. It's light-weight, fully open-source (distributed under Apache license) and written in Java. It allows to model, execute and monitor business processes, throughout their life cycle.

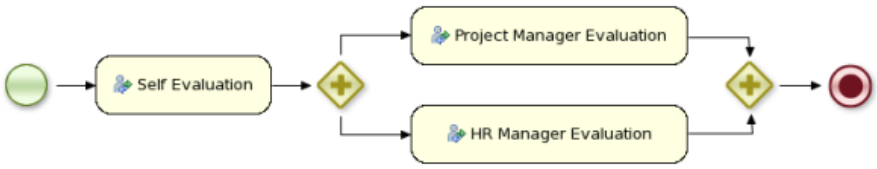

Figure 2.1: Example jBPM process as shown in [10]

A business process allows to model various business goals by describing the steps that need to be executed to achieve that goal and the order, using a flow chart. This greatly improves the visibility and agility of a business logic. jBPM focuses on executable business process, which are business processes that contain enough detail so they can actually be executed on a BPM engine. Executable business processes bridge the gap between business users and developers as they are higher-level and use domain-specific concepts that are understood by business users but can also be executed directly.

The core of jBPM is a light-weight, extensible workflow engine written in pure Java that allows to execute business processes using the latest BPMN 2.0 specification. It can run in any Java environment, embedded in an application or as a service.

BPM makes the bridge between business analysts, developers and end users, by offering process management features and tools in a way that both

business users and developers like it. Domain-specific nodes can be plugged into the palette, making the processes more easily understood by business users.

jBPM supports adaptive and dynamic processes that require flexibility to model complex, real-life situations that cannot easily be described using a rigid process. jBPM is also not just an isolated process engine. Complex business logic can be modeled as a combination of business processes with business rules and complex event processing. jBPM can be combined with the Drools project to support one unified environment that integrates these paradigms where business logic can be modelled as a combination of processes, rules and events.

Apart from the core engine itself, there are a few optional components that can be used. Eclipse-based or web-based business process designer and a management console for the execution engine.

2.5.1 jBPM Core Engine

The core jBPM engine is the heart of the jBPM project. It's a light-weight workflow engine that executes business processes. It can be embedded as a part of an application or deployed as a service. It's most important features are:

- Solid, stable core engine for executing process instances
- Native support for the latest BPMN 2.0 specification for modeling and executing business processes
- Strong focus on performance and scalability
- Light-weight (can be deployed on almost any device that supports a simple Java Runtime Environment, does not require any web container at all)
- (Optional) pluggable persistence with a default JPA implementation
- Pluggable transaction support with a default JTA implementation
- Implemented as a generic process engine, so it can be extended to support new node types or other process languages
- Listeners to be notified of various events
- Ability to migrate running process instances to a new version of their process definition

The core engine can also be integrated with a few other (independent) core services:

- The human task service can be used to manage human tasks when human actors need to participate in the process. It is fully pluggable and the default implementation is based on the WS-HumanTask specification and manages the life cycle of the tasks, task lists, task forms and some more advanced features like escalation, delegation, rule-based assignments, etc.

- The history log can store all information about the execution of all the processes on the engine. This is necessary if access to historic information is needed as runtime persistence only stores the current state of all active process instances. The history log can be used to store all current and historic state of active and completed process instances.

  It can be used to query for any information related to the execution of process instances, for monitoring, analysis, etc.

CHAPTER 3

Přiříz architecture and requirements

Previous chapters should provide enough information to understand basic principles of a RESTful service and benefits of using BPM. Discussion about the two mentioned will become more concrete from now on, because admission process will be taken into account.

As a result of this master's thesis a working application is implemented. Further referred as **RESTful API** and it integrates both technologies.

This chapter describes requirements for application functionality and its role in the architecture of the whole project currently running at CTU FIT.

3.1 Faculty Information System (FIS)

The project covers activities related to information system development CTU. It is quite complicated and long term project involving more than dozen people in various roles. RESTful API with two other applications directly interconnected is just a part of it and together they form **Přiříz** component 3.1.

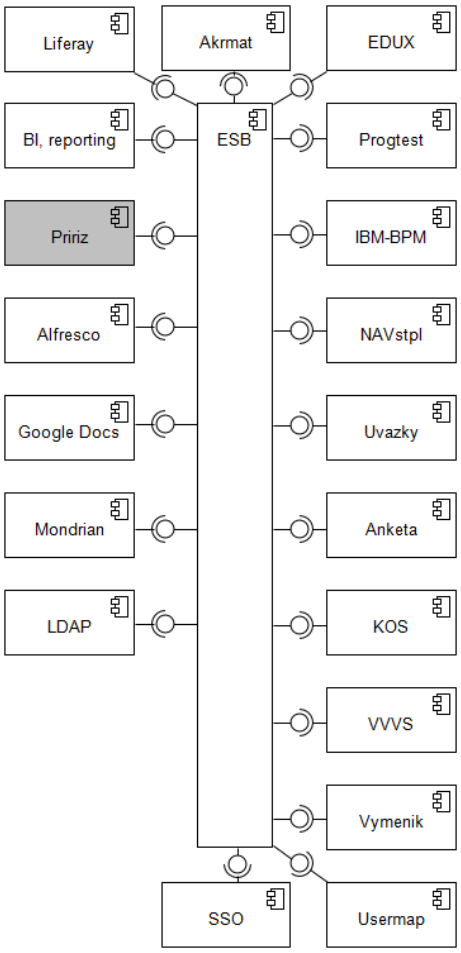

Figure 3.1: FIS architecture

3.2 Catalogue of requirements

This section lists requirements for the RESTful API.

3.2.1 Functional requirements

Functional requirements define which services should system provide.

- F00 Admission import
  data import from e-admission, on-line CTU form
- F01 Admission evidence
  evidence of valid admissions

- F01.1 Admission detail

  detailed information about admission and admissioner

- F01.2 Admission edit

  update of allowed admission data

- F01.3 Password reset

  allow password reset/recovery to a valid user account related to the admission

- F02 Term evidence

  entrance exam term evidence

- F02.1 Term management

  entrance exam term management

- F02.2 Enrollment term management

  enrollment term of accepted admissioners management

- F02.3 Term registrations evidence

  Entrance exam or enrollment term registrations of admissionsers evidence

- F03 Statistics

  various statistics of this year's admission process

- F04 User management

  password change

- F05 Admission state

  view current state of admission

- F05.1 Entrance exam registration

  allow an admissioner to book entrance exam term

- F05.2 Entrance exam apology

  allow an admissioner to apologise from the entrance exam registration

- F05.3 Enrollment registration

  allow an admissioner to book enrollment term

- F05.4 Enrollment apology

  allow an admissioner to apologise from the enrollment registration

- F06 Admission process management

  allow management of an admission during admission process

- F06.1 Send e-mails

  allow sending of informative e-mail

- F06.2 User action processing

  allow processing of user actions during admission process

- F07 File number administration

  allow file number assignment to documents communicated with an admissioner

- F08 Document creation

  allow document creation of various types for an admissioner

3.2.2 Non-functional requirements

Non-functional requirements do not directly define system functionality, but describe constraints or general properties.

- NF01 User action processing

  jBPM engine requires local TaskService server to handle user actions in process

- NF02 E-mail server configuration

  jBPM engine requires properly configured mail sender to be able to use CTU's mail server

- NF03 Technologies

  use the following for implementation: JEE, Apache Maven, jBPM, Spring Framework, Spring Roo, JPA

- NF04 System and web service security

  system and web services will be secured

- NF04.1 User roles and permissions

  security will be handled by user roles and permissions

- NF05 MySQL database

  primary DBMS will be MySQL

- NF06 Server Tomcat

  RESTful API will be able run on Apache Tomcat servlet container

- NF07 Performance under load

  system must handle 250 concurrent users and 2500 total users

- NF07.1 Scaling

  system must be able to split load among multiple instances

During the development process several changes were made to the catalogue of requirements. RESTful API team and UI team switched responsibilities for F07, F08. F03 was descoped but can be implemented with relatively small effort when needed.

NF07.1 is a matter of infrastructure. While this work was being created, only one virtual server was available for the whole Přiříz project and makes no sense to configure multiple instances on a single machine.

3.3 Who and how will use RESTful API?

I already mentioned that RESTful API will be consumed by two different teams 3.3. This is their simplified requirements catalogue - functional requirements only:

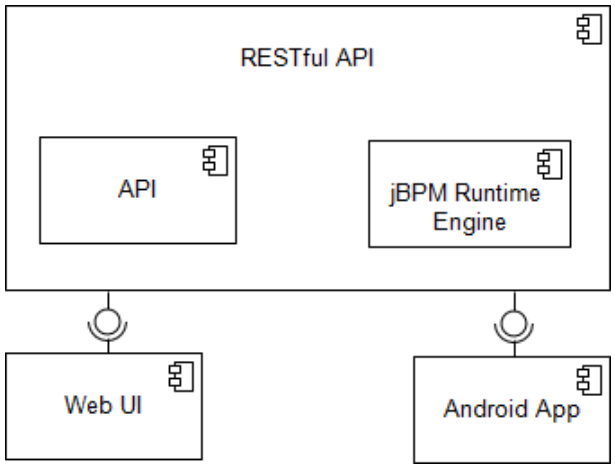

Figure 3.2: Přiříz project components

- Android team

  Responsible for native Android application development. Its tasks are:

  - allow to log-in in various user roles
  - log-out
  - barcode identification

  - view admission/admissioner information
  - save admission result - entrance exam score
  - take a picture of a document and upload it to server

- UI team

  Implements web interface. This allows admissioners and CTU FIT's employees to interact with admissions and manage them during admission process.

  - Admission/Admissioner
    * log-in/log-out
    * view personal information
    * book a term (entrance exam, enrollment)
    * apologise from a term registration
    * change/reset password
  - Employee (Study Department staff, sub dean, Faculty departments staff)
    * log-in/log-out
    * list admissions, filter
    * view admission detail
    * edit/delete admission
    * reset user's password by admission
    * view terms (entrance exam, enrollment)
    * create/edit/delete terms
    * import admission data from KOmponenta Studium, Study Information System at CTU (KOS)
    * view study programs
    * edit/delete study programme
    * confirm admissioner's attendance at entrance exam or enrollment
    * accept or decline admissioner's apology
    * edit properties of an admission process
    * view statistics
    * change password

Although UI team has quite a long list of task when compared to Android's, they practically overlap. This is why both (and for future all) teams are going to share the same API and RESTful API will handle all consumers the same way.

All requirements from the list above, but those platform specific, will use RESTful API as a backend and should not store any persistent data on their own. Platform specific means that they must store runtime data on their own. An example may be HTTP session or other temporaries.

The task for RESTful API team is to expose a public API, which will satisfy requirements from the list above.

CHAPTER 4

Chosen technologies

This chapter describes pros and cons of available tools, frameworks and architectural patterns, that may be used and explains why and which technologies I finally selected.

One of the non-functional requirements from the previous chapter 3.2.2 talks about technologies, which should be used. The truth is that NF03 was back-added after discussion described in this chapter.

4.1 REST vs. SOAP

Although implementing a RESTful API is one of the main topics of this master's thesis, it is good to compare other possibilities too. Currently the most commonly used technology to build SOA in addition to REST is SOAP. So what are the differences between them? Why should not I use SOAP, when thousands of enterprise systems are using it?

Sometimes people talk about SOAP like it was something that is deprecated, or even dead and REST is its successor and is much better and modern. This is not true and not even part of it. REST is no revolution in SOA, but rather an evolution. SOAP has its place when it comes to a question of implementing services or APIs and so does REST. The main difference is: **SOAP** is aimed for **server-server** communication and **REST** is more suitable for **client-server** communication.

Let's forget about the RESTful API requirement and make it just an API. What are the pros and cons of REST, resp. SOAP if I could choose one or the another on my own? The only things that I have to keep in mind: I'm implementing an API for admission processing and I have two different API consumers using two various platforms 3.3.

	REST	SOAP
Data Format	XML, JSON, YAML, ...	XML
Transport	agnostic, but very tightly coupled with HTTP, unlikely to use anything else	agnostic
Error handling	implementation specific	built in
Primary use	client-server directly	server-server, possibly via mediators
CRUD[a]	HTTP methods	implementation specific
Interface description	text description, XSD for XML representation	standardized WSDL, optional XSD
Tools availability	lacking, partial	outstanding
Security	HTTPS, implementation specific	built in, WS-Security
Pros	Lightweight Space saving formats Easy to learn	Standards Extensible Tool support Type checking Rigid
Cons	HTTP dependency Lacking standards Lacking tools Assumes point-to-point use	Bloating data format Complexity and learning curve

[a]Create Read Update Delete

Table 4.1: REST vs. SOAP properties

Does not look very good for REST from the overview above. Using simple math, it has less pros and more cons. But does it mean that SOAP should be chosen for API that Přiříz project needs?

Android team uses a mobile phone device, which is definitely a client. What about the Web interface? I am not sure about the implementation, but from my point of view, it should be a thin client, which stores only necessary runtime data for its own needs. No need to synchronize any other data, just consume and present. This means **two clients** will be using the API.

Web interface will be deployed on its own application server somewhere inside the CTU's infrastructure and will use fast network connection. On the other hand, Android device will be able to use some wireless fast network -

best case. But what if the mobile application will be forced to rely on a slower carrier network? EDGE - worst case. The API should then try to save as much traffic as it can. XML full of namespaces that **SOAP** happily uses is **not** the **right** way, though.

HTTP dependency is not a problem, I would use SOAP over HTTP anyway. Lack of standards is a matter of good design. This is why I have to be very careful when exposing new functions or data models. It does not mean that this issue would not effect SOAP design.

Lack of tools is a thing that bothers me a bit. If I agree on standardized approach with other teams and we involve a bit of communication, hopefully we can get over this one.

So far for the cons of REST. Which pros of SOAP will I miss, if I decided to use REST? Just the tool support.

To sum up, RESTful API would win and therefore I am happy with the original requirement.

4.2 BPEL vs. BPMN

When I started to collection information about BPM and its standards, I found many discussions about older BPEL vs. relatively new BPMN, currently available in BPMN 2.0 specification.

[4] says that the only standard worth considering is BPEL. This information is a few years old and by the time a another player has shown up. BPMN 2.0 became a solid competitor to BPEL.

[9] BPEL and BPMN are both „languages“ or „notations“ for describing and executing business processes. Both are open standards. Most business process engines will support one or the other of these languages.

It turns out that BPEL is really well suited to modeling some kinds of processes and BPMN is really well suited to modeling other kinds of processes. Of course there is a pretty significant overlap where either will do a great job.

[3] BPMN 2.0 is now a business model that can be executed after implementation details are added. BPMN favors the business user, even though a developer can „refine with execution semantics“ to make it executable. It is graph based, and incorporates user swim lanes, which makes it effective for modelling end to end business processes.

BPMN 2.0 introduces a standardized file format (previously is was proprietary or converted to XPDL). BPMN looks like a version of BPEL where the assigns are tucked away into other activities to clean up the diagram.

BPEL's nature is still service orchestration, and will be great for building composite services and integrating with applications. BPEL will still probably be the choice for developers, where BPMN will be good for the pure decision layer and **Human Task interaction**.

[7, p. 1] The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation. Another goal is to ensure that XML languages designed for the execution of **business processes**, such as Web Services Business Process Execution Language (WSBPEL), **can be visualized** with a business-oriented notation.

Human Task interaction is exactly what admission processing is looking for and visualization of the process is a must. It is expected that further modifications of business process model will be performed by non-developers. Therefore an user friendly visual modelling tool is required. That is why BPMN 2.0 looks like a good choice. Luckily jBPM, which was chosen as a primary technology for this master's thesis, fully implements BPMN 2.0 standard.

4.3 JAX-RS implementation

4.4 Spring Core, MVC, Security, ...

4.5 Spring Roo

Bibliography

[1] Burke, B.: *Restful Java with Jax-RS*. O'Reilly Media, 2009.

[2] Compaq/W3C; W3C/MIT; Xerox; etc.: Request for Comments: 2616. June 1999. Available at WWW: <http://www.ietf.org/rfc/rfc2616.txt>

[3] CONSULTING, M.: BPMN 2.0 vs BPEL (Oracle BPM VS SOA Suite). September 2010. Available at WWW: <http://www.mandsconsulting.com/bpmn-20-vs-bpel-oracle-bpm-vs-soa-suite>

[4] Havey, M.: *Essential Business Process Modeling*. O'Reilly Media, 2005.

[5] Lab, L.: PATCH Method for HTTP draft-dusseault-http-patch-16. November 2009. Available at WWW: <http://tools.ietf.org/html/draft-dusseault-http-patch-16>

[6] Microsoft: MSDN 2.2.4.1 PATCH/MERGE. Available at WWW: <http://msdn.microsoft.com/en-us/library/dd541276(v=prot.10).aspx>

[7] OMG: Business Process Model and Notation (BPMN). January 2011. Available at WWW: <http://www.omg.org/spec/BPMN/2.0>

[8] RFC 2616 Fielding, e. a.: part of Hypertext Transfer Protocol – HTTP/1.1. Available at WWW: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

[9] Shepherd, P.: Architecture Standards – BPMN vs. BPEL for Business Process Management. March 2011. Available at WWW: <https://blogs.oracle.com/enterprisearchitecture/entry/architecture$_s tandards_b pmn_v s$>

[10] jBPM team, T.: jBPM User Guide. May 2012. Available at WWW: <http://docs.jboss.org/jbpm/v5.3/userguide/>

[11] Xerox: Request for Comments: 2396. August 1998. Available at WWW: <http://www.ietf.org/rfc/rfc2396.txt>

APPENDIX A

Content of CD

readme.txt	- the file with CD content description
data/	- the data files directory
graphs/	- the directory of graphs of experiments
*.eps	- the B/W graphs in PS format
*.png	- the color graphs in PNG format
*.dat	- the graphs data files
exe/	- the directory with executable WBDCM program
wbdcm	- the WBDCM program executable (UNIX)
wbdcm.exe	- the WBDCM program executable (MS Windows)
src/	- the directory of source codes
wbdcm/	- the directory of WBDCM program
Makefile	- the makefile of WBDCM program (UNIX)
thesis/	- the directory of LaTeX source codes of the thesis
figures/	- the thesis figures directory
*.eps	- the figures in PS format
*.pdf	- the figures in PDF format
*.tex	- the LaTeX source code files of the thesis
text/	- the thesis directory
thesis.pdf	- the Diploma thesis in PDF format
thesis.ps	- the Diploma thesis in PS format