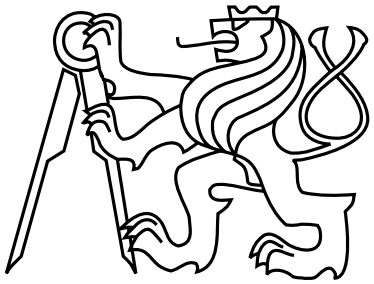
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Master's Thesis

Semantic Data Analysis and Visualization of User Interactions

Bc. Michal Švácha

Supervisor: Ing. Ivo Malý Ph.D.

Study Programme: Open Informatics

Field of Study: Software Engineering

May 18, 2016

Aknowledgements

I would like to thank dearly to my cat, dog and goldfish I never had. You were a true inspiration to me. With you, I would have never made this document happen. Also, my PlayStation 3 proved to be an amazing tool to keep me sane while writing this thesis. Thank you SONY.

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 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on May 27, 2016 .....

Abstract

The current status quo in data collection is to collect everything that is available. This approach has given rise to a trend in the last couple years - "Big Data". Data that is inconveniently large for processing, interpretation and inference. When a company decides to leverage big data, it usually ends up having too much information and no real business value. Each department uses different domain vocabularies and there is therefore no synchronization or understanding. This master's thesis is trying to take the opportunity of this disorder and connect the endpoints together while leveraging big data to provide an end-to-end solution.

The focal part of this thesis is the focus on user interactions - the collection of data from mobile applications. Before such collection can even happen, the interactions must be defined - the "what", "when" and "why". Starting with management, over to architecture and engineering to interpreting results as the destination - uniting all steps to form a bigger picture.

Abstrakt

Momentální status quo ve sběru dat je sbírání a ukládání všeho, co je k dispozici. Tento přístup dal vzniknout trendu posledních let - "Velká data". Tedy data, která jsou nepohodlně objemná pro zpracování, výklad, a odvozování závěrů. Když se společnost rozhodne, že chce využít velká data, dopadne to většinou tak, že má příliš mnoho informací bez žádné reálné hodnoty. Každé oddělení používá vlastní doménové názvosloví a tudíž chybí synchronizace a porozumění. Tato diplomová práce se snaží využít této příležitosti neuspořádanosti pro spojení všech konců dohromady a vytvořit tak komplexní řešení.

Těžištěm práce je zaměření se na uživatelské interakce - sběr dat z mobilních aplikací. Než nějaký sběr vůbec nastane, je třeba mít interakce definované - tedy "co", "kdy" a "proč". Počínaje projektovým managementem, přes architekturu a softwarové inženýrství až k interpretaci výsledků jako konečným bodem - spojení všech kroků k vytvoření uceleného náhledu.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the lean/agile product development, it is necessary to have a formalized user feedback loops in place, to measure the product performance against various (quantitative) metrics. Such feedback loops are obtaining the information and statistics regarding user engagement and interaction with the developed product. Are the users using it the way the creator imagined it or did they find any other means of utilizing it? What stops the users from doing the task they intended? Do they get everything they need and at the same time does the creator get what was expected? In other words - are the dominant means of usage incentive compatible for the users?

Not only that the current solutions for gathering such user data for both quantitative and qualitative metrics work out-of-the-box with low-level semantics only (everything is a general activity on a general resource), but also tend to run on somebody else's servers. What if the product developer is in a highly regulated market, such as the pharmaceuticals, and has to own all their users' data? How can the current solutions' space be utilized and tweaked in order to fit in such schema?

The most problematic issue in large corporations is the disconnectivity of the data. It may physically be all there, however, nobody knows what and how should it be connected in order for it to make sense and drive value. Do I have good data or do I just have petabytes of useless log trace? Am I gathering what the application was intended to do or am I only filling the database with irrelevant garbage information. Most importantly, though - Am I gathering the information I need in a consistent fashion that corresponds to the domain of the shareholders?

Let's draw an analogy here.

Example

1. Top management comes with a need to sell more products on mobile devices.
2. Project management team takes over and breaks it down to user-stories, tasks etc.
3. Project management team asks the UX team, the mobile applications team and the backend team to perform their tasks in order to bring the product to life.

4. By the time the product development is spread in three different teams, it is very much likely, that each team creates their own jargon for every activity performed in the product.
    - The UX team is driven by the larger picture so they tend to refer to objects in a highly abstract way.
    - The backend team is driven by the inner processes happening between the database and the REST API, so they speak their precise technical language.
    - The mobile applications team is somewhere between, but never really aligned with either of the other teams.
5. When the product is delivered, the project management team has a really hard time translating reported results into the top management's primary goals. Plus - the larger the teams are, the worse the situation actually gets.

This example isn't a problem in project management itself. The company can use the best project management tools on the market and have the smartest people working on their teams and still encounter this problem on a daily basis. This is a larger problem of domain disconnectivity.

1.0.1 Personal interviews

In order to verify the needs I have conducted several interviews with leaders of various departments. These are their reactions to what bothers them about product monitoring during development phase:

1. Associate Director, Applied Technology

    "The real problem I see is the fact, that all the information I need is on somebody else's server. We can't store any sensitive, let alone confidential information *somewhere* with some random vendor. It's actually illegal in some countries. Unfortunately, sometimes sensitive data is exactly what we need to obtain from the applications to make an informed decision."

2. Associate Director, Mobile and Web

    "Our needs for tracking KPIs are variable throughout the time and unfortunately the current tools we use are quite inflexible. Because we are in a regulated market, each change that requires new build of an application takes a longer period of time. And time *is* money."

3. Mobile Application Development Lead

    "I have noticed that the one of the biggest obstacles is how should we name what we measure. I have no vocabulary to help me during the development. The only thing we have is a robust Google Analytics toolkit that only allows us to gather low-level actions. We can log that a button was pressed, but what do we name such action? "Button pressed"?"

4. UX Lead

> "Our team looks at the high level needs. We are trying to make the user activities in an application as smooth as possible. When we design a low-fidelity prototype, we know what we want to measure. Having the opportunity to add a high-level KPI would help us a lot in order to gather feedback for our prototypes. We are not programmers, we don't know how to add it to the code, but I would love the idea of including what to measure along with the prototype."

Scope of work

First, I will present the context of agile software development in a regulated market. Then I will take a look at the current tracking solutions being used in mobile development (focus is on iOS development, but most of the tools are multiplatform solutions) - I will examine and analyze their strengths and weaknesses.

I will further propose a workflow to fit the needs of a larger company with multiple departments, operating in a regulated market. I will utilize existing tools and build on top of them in order to drive value without reinventing the wheel couple times.

Last I will develop a working PoC (Proof of Concept), verify its functions by user testing and further discuss with the stakeholders again whether or not it is the correct path to uniting all departments and connecting the dots in the data.

Chapter 2

Analysis

2.1 Opportunity Assessment

1. *What problem are we trying to solve?*

   The problem is that no commercially successful platform allows storage of user data on custom servers. In a regulatory market it is vital to have such option. Also no higher semantic analysis other than defining in-code customizations is provided. Enabling alignment of business language and user interaction reporting is a key part to success. The lack of interchangeability tends to end up with a vendor lock-in.

2. *For whom do we solve that problem?*

   For all departments in the company that are participating in product development, top to bottom. From setting a business goal to defining the needs, while also helping the developers and the designers.

3. *How will we measure success?*

   Having gathered data that is securely placed on custom servers. Data that is united and connected by shared domain vocabulary, ready to be analyzed and visualized in order to draw conclusions.

2.2 Regulated environment constraints

Regulated markets, especially pharmaceuticals have multiple rules that need to be carefully followed in order to be allowed to use new IT products. Not only is important how much value does the final product bring to the end user, but also how data storage is handled and how prone it is to exploitations and attacks.

From development process point of view it is equally important to follow specific guidelines and processes during the development phase. Each step has to be carefully documented and approved by specific audit department. For that reason, most of the big pharmaceutical companies use the waterfall model - SDLC (= Software Development Life Cycle), which enables companies to follow specific steps in order to get their software product certified.

This all hassle is not for the sake of bureaucracy, it is to protect the customers and increases the level of tracability of a problem, should one ever occur. After all, it is their private data that goes on the server, so it is imperative that it is safe.

2.3 Existing Tools

2.3.1 Google Analytics

Google Analytics is the by far most popular and widely used framework for monitoring user interactions in applications. Reports everything the developer wishes to. By default it doesn't report anything - the tool has to be activated on application start and then actions need to be wired to the framework. Action can be either wired up after certain custom occurrence has appeared (pressed button, refreshing data) or simply an identifier can be set to an actionable item (button) - then whatever happens with that item, gets reported.

The dashboard website is very detailed and responsive. All data is nicely visualized in graphs and corresponds well with the whole GA ecosystem. Higher order semantic interpretations are missing, though.

The source code is closed source and all of the statistics run ot Google servers. Data is accessible via REST API, but registration is required. Single user account is free, enterprise account is paid for. Enterprise account does NOT include an opt-out from Google servers.

2.3.2 Fabric (formerly Crashlytics)

Crashlytics was also a star framework. Acquired by Twitter, it is now a vital part all purpose platform - Fabric. Fabric is aside from a reporting tool a full featured developer platform used for variety of tasks and obstacles a developer may face - even beta version distribution that has always been a problem for iOS developers. The Crashlytics reporting has been taken a step further and is not only about reporting crashes. It also reports overall statistics, like Google. One nice feature Fabric has is by acquiring AppSee - a way of visualizing user movement in the app through the usage data. A video of steps users take in their app gives the developer a new perspective on how the app is used.

The source code is closed source and all of the statistics run ot Twitter servers. Data is NOT accessible. The whole platform is free to use. Registration is required along with installation of a custom program to install the framework parts in existing projects.

2.3.3 App Pulse (formerly Pronq)

App Pulse is a "new feature by acquisition" - acquired by HP in 2014, it is now part of the portfolio of the new Hewlett Packard Enterprise. It lets users try out their 30-day trial and then charges for everything (no free version).

App Pulse is very different from GA in a way that it reports everything at all times. The usage is fairly low - tens of kilobytes per week, but it is very thorough. Screen time, actions, movements - it is all there. No setup is required for the start. The SDK they supply simply has to be dragged and dropped in Xcode project and then it starts working out-of-the-box.

The only issue is the need to have consistent naming of all views, labels, buttons etc. - as it does everything on its own, without hooking up the actionable items to the framework manually, it can be hard to determine which button was which.

The tools are closed source and all of the statistics run on HP servers. No API is provided.

2.3.4 Crittercism

Crittercism doesn't stand out from previously mentioned tools - has their own servers, SDK and works seemlessly. Has some more benchmarking than others, and seems very enterprise oriented - they enable 3rd party API integration into their system to see performance of other APIs used in the application to really find what can be the bottleneck of the app's performance.

The tools are closed source and all of the statistics run on their servers. API is provided.

2.3.5 New Relic

New Relic somewhat differs in a sense that as the only platform there is a mention on their website about "specific needs" - maybe custom server can be provided. Otherwise it is the same strategy - SDK installed in every app and statistics gets reported periodically. Nice alerting system is optionally provided - when crash occurs, web hook to ticketing system can be defined to streamline bug reports.

The SDKs are open source, the analytics runs on their servers (possibly 'not only'). No API is provided.

2.3.6 Apple

The last isn't considered framework, but it should be noticed. As companies fight for data from mobile applications - such as Twitter, who gives out their platform free for everybody, naturally the platform owners strive for keeping all that precious data for themselves. Apple announced at WWDC 2015 new iTunes Connect portal redesign and along with it also a new feature - App Analytics. It is fairly thorough in means of usage, downloads, screen time etc, but overall reporting is still very high level and really far from the code. It seems like it is not meant to be a developer tool at all, because there is no in depth code reporting. There are crashes reported, but not very detailed compared to Fabric.

These statistics are provided to every single developer of iOS apps for free on the iTunes Connect website. There is no framework and naturally Apple keeps all of the data for themselves.

Conclusion on tools

Why are they bad etc etc.

Chapter 3

Design

3.1 System Architecture

PICTURE

Description

TBD depending on the picture

Components

3.1.1 Issue Tracker

JIRA is ...

3.1.2 Semantic Data Manager

This is the most crucial component of the whole solution - connecting JIRA and tracking capabilities. The main task is to obtain actionable items from JIRA and create configurations for application engineers to include in the source code. These configurations should be persisted for the sake of reproducibility. Persistence shouldn't be applied to any metadata (detailed measurement description) as it is subject to change in JIRA. Every issue/ticket has a finite unique ID and that should be the only persisted piece of data.

Connecting to JIRA will be handled via its JIRA Agile REST API. It should automatically retrieve a list of all projects and their subsequent issues/tickets that may or may not contain specific metadata regarding fine-grained measurement demands.

SDM will run as a stand-alone microservice and provide UI for easy measurement configuration, but also REST API, should the UI ever be replaced.

3.1.3 Tracking Engine

As a tracking engine, anything that provides API for data storage/retrieval is good enough. Google Analytics is a good feasible option, but for reasons listed in previous chapter (especially privacy), it is more convenient to use own tracking solution.

3.1.4 Tracked Device

This can be any kind of iOS mobile device - iPhone, iPad, Apple Watch or Apple TV. The key part is, that the data that is being sent to the tracking engine is in sync with the data that the semantic data server collected from JIRA.

3.1.5 Statistical Front-end

This component is the most visible one, because it interprets the collected data.

Analýza a návrh implementace (včetně diskuse různých alternativ a volby implementačního prostředí).

Chapter 4

Implementation

4.1 Deployment

// Pretty picture

Everything is deployed in the Amazon Web Services (= AWS) cloud environment.

Semantic Data Manager is deployed in AWS Elastic Beanstalk (= EB), which is ... Database runs on managed Amazon RDS (Relational Database Service), which is fast, secure and scalable deployment of database engine. The default database engine is MySQL.

Tracking Engine is also deployed in AWS EB but uses Aurora as database engine. Aurora is ... and has ... configuration, allowing to have really high traffic and maintain its speed and robustness.

Statistical app ...

4.2 Issue Tracker

The issue tracker used in my work environment is JIRA[1] by Atlassian. It is a standard project management tool providing bug tracking, issue tracking and many other functions. It is conveniently synergistic with other Atlassian tools such as Confluence (for documentation and wiki) and Bitbucket (formerly Stash), which is a server for version control (Mercurial and Git). The advantage is that all these three components are deployed in a Virtual Private Cloud (= VPC) environment - Amazon Web Services. Thus, as a programmer I have fairly easy access to its internal API without being afraid to leak data where it shouldn't.

4.2.1 Knowledge structure

Every company uses a little different way of structuring their knowledge in their project management tools. In my situation, the structure was as follows:

[1] No abbreviation here. It is short for GOJIRA, which is Godzilla in Japanese. Rumor has it that it is because the main competing product is Bugzilla.

4.2.1.1 Teams

Everybody is distributed into teams in various departments. While departments are important in the company structure, in the project management, teams are more relevant. Every team has its own Project Board (either Scrum or Kanban) that reflects the state of the team in time - how much work has been done, what will be done and what's currently being done.

4.2.1.2 Projects

Each team works on their own projects. Often times, collaboration does occur, but the owner of the project is still the team. If other person collaborates with other team, it is reflected in his/her work statistics (storypoints, hours spent), but the ownership stays within the scope of the project and thus the team as well.

4.2.1.3 Issues

Issue is the most granular entity in the system. It defines a step or set of steps to be performed and is usually assigned to one person. Workflows for reviewing etc. vary from team to team. There is a possibility to break down the work on single issue (if it's especially large one) to sub-tasks, but those are not given IDs, so they are not unique per se. They only serve for clarity and visibility of work being done during the day. When all sub-tasks are finished, the whole issue is finished. That is the state reflected by the API.

4.2.2 API

JIRA REST API is quite a mess, which is because Atlassian didn't develop all their products from scratch (Bitbucket was acquired) and it is still visible that the usage isn't seamless. In order to access Teams, Projects and Issues, two API endpoints have to be used:

1. JIRA REST API
2. JIRA REST AGILE

Both of them are APIs (= Application Programming Interface), but I will use their names to distinguish one from another.Both are similar, but also slightly different from each other.

To illustrate the subtle differences that drive any software engineer mad:

- When JIRA REST API is used to obtain the issues, the resulting array uses pagination, because there could be a lot of issues and loading them all at once could take a significant amount of time. In order to determine whether the array I have is final, a parameter "total" is present in the response. This parameter tells how many issues in total there are. In order to load the whole list, it is necessary to keep track how many there are, and how many are left on the stack.

- When JIRA REST AGILE is used to obtain the teams, the resulting array also uses pagination. In order to determine whether the array I have is final, a parameter called "isLast" is present in the response, having, surprisingly, a boolean value true/false. Obviously, when the value is false, one has to load the next page with the last index that came before.

There are plenty of these little surprises that are so easily breakable with any update of the whole system. I honestly do not know, why it isn't the top priority for Atlassian to unite their APIs.

What struck me most though, is the absence of OAuth or Token-based communication. Every query is done via basic auth. While for development it is fine as it allowed me to quickly prototype on top of the API without the need to develop a complex token manager, for production it is quite inconvenient. Even though SSL certificates are all valid and in place, it simply is a terrible architectural choice to not have a proper way to authenticate other applications using the APIs.

4.2.2.1 JQL

JQL stands for JIRA Query Language[2]. It enables the API user to query the JIRA knowledge graph and extract information.

Figure 4.1: JQL syntax

1. **Field** - Fields are different types of information in the system. JIRA fields include priority, fixVersion, issue type, etc.
2. **Operator** - Operators are the heart of the query. They relate the field to the value. Common operators include equals (=), not equals (!=), less than (<), etc.
3. **Value** – Values are the actual data in the query. They are usually the item for which we are looking.
4. **Keyword** – Keywords are specific words in the language that have special meaning. In this post we will be focused on AND and OR.

I used JQL in order to get all issues for a certain project:

```
"https://jiraURL/issues/search?jql=project=SAUI"
```

Here I used simple query to search all issues where the project is SAUI (= Semantic Analysis of User Interactions). All URL encoders handle the double "= =" and it has never happened to me, that it would encode the parameters badly.

It can obviously be even more powerful, but I was glad it helped me easily get what I needed.

4.2.2.2 Methods used

All communication is handled via HTTP GET and all responses are in JSON format. Cross-site request forgery (= CSRF/XSRF) token system is disabled.

1. To get all teams, method /**board** has to be called on JIRA REST AGILE.
2. To get all projects, method /**projects** has to be called on JIRA REST API.
3. To get all issues, method /**search** with JQL query has to be called on JIRA REST API.

Interestingly enough, even though, there are two API endpoints, the data is connected, so no further processing was necessary. It is important to note, that the responses are **very** verbose and it is possible to tell in the query to the server not to send some fields back.

4.2.3 DSL

In order to track more than just the beginning and the end of the workflow defined in the scope of the issue, it is necessary to give the project managers a way to define certain observable keywords to pay attention to. Every issue has a field called "description". Usually it contains some human readable set of instructions. Why not piggyback on that and give it just enough structure to make it also computer readable?

I came up with a simple solution - add keyword "WATCH:" on new line and describe what to observe. Parsing is done line by line where the code searches the line for "WATCH:" (case insensitive) and extracts whatever follows until the end of line or occurrence of another "WATCH:". In order not to make it complex, end of line is the end of any description, it does not carry over to the next line.

4.2.3.1 Examples

Here are some examples how the parser for the DSL works. Validation of this technique will be covered in the Testing chapter.

Example 1 - Success

```
As a user, I want to be able to list all projects in the mobile app currently
    being tracked along with the number of tracked versions in the tracking system.

WATCH: Number projects expanded to the highest detail
WATCH: Filters used to extract information
```

This succeeds perfectly, as it parses everything without any hassle.

Example 2 - Success

```
As a user, I want to be able to list all projects in the mobile app currently
    being tracked along with the number of tracked versions in the tracking system.
Watch:    Number projects expanded to the highest detail
Watch:    Filters used to extract information
```

This succeeds too, because the search is case insensitive and after the search, white spaces are extracted, so the result is the same as in the previous example.

Example 3 - Semi-success

```
As a user, I want to be able to list all projects in the mobile app currently
    being tracked along with the number of tracked versions in the tracking system.

WATCH: Number projects expanded to the highest detail, Filters used to extract
    information
```

This is a semi-success, almost a failure, but it still yields all the information that the user wanted. It just isn't nicely separated and would need some changes. The error is visible to the programmer and is easy to fix.

Example 4 - Failure

```
As a user, I want to be able to list all projects in the mobile app currently
    being tracked along with the number of tracked versions in the tracking
    system. WATCH: Number projects expanded to the highest detail. WATCH: Filters
    used to extract information
```

This yields only one result - "Number projects expanded to the highest detail.". While it seems like it is a good solution, the fact that it seems that way is unfortunately the worst thing about it - because it is hard to discover that there is an error. The programmer sees one observable action item and doesn't see that some got lost during the process, because it was all on one line.

4.3 Semantic Data Manager

The central part of the project should be robust and reliable. For that reason I chose Java as the main technology. For convenience and standardization of the code-base I opted for Spring Boot framework to help me with bootstrapping the heavy work (scheduling, threading, persistence etc.).

4.3.1 Spring Boot

I first tried to use play2 framework for educational purposes, but I encountered too many obstacles deploying play2 application to AWS:

- It does not support WAR packaging.[2]
- It is not possible to run play2 packages (packaged by Activator tool) on Tomcat Server.
- It comes with its own Netty Server, which is really clumsy to set up in AWS environment.

All three combined resulted in inability to synchronize the play2 application on port 9000 and NGINX running on port 5000. Unfortunately Netty Server does not support compile-time port configuration and NGINX does not support running Activator to set up the port during run-time, so I had to drop the idea of using play2 as I was simply unable to deploy the application. After researching and discussing with my peers and coworkers I looked up Spring MVC and stumbled upon Spring Boot, also recommended by my classmate. I tried few sample apps and found out it supports WAR packaging, runs natively on Tomcat and comes with almost the same perks like play2. I was ready to give it a try.

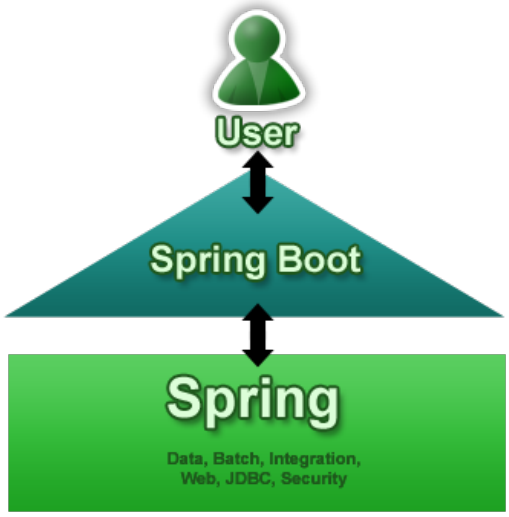


Figure 4.2: Spring Boot architecture

"Spring Boot aims to make it easy to create Spring-powered, production-grade applications and services with minimum fuss. It takes an opinionated view of the Spring platform so that new and existing users can quickly get to the bits they need."[5]

Primary goals of Spring Boot are[4]:

- Provide a radically faster and widely accessible getting started experience for all Spring development.
- Be opinionated out of the box, but get out of the way quickly as requirements start to diverge from the defaults.

[2]There is an unofficial tool that packages the code in a WAR file, but it is not recommended for production environment. Being constrained by highly regulated market, something that already says that it is not production ready is an instant "No thanks".

- Provide a range of non-functional features that are common to large classes of projects (e.g. embedded servers, security, metrics, health checks, externalized configuration).
- Absolutely no code generation and no requirement for XML configuration.

The advantages seemed to be strong, development environment was convenient (IntelliJ IDEA native support) and after some validation with Amazon support regarding AWS EB deployment, I was confident this would be a good choice.

// H2 DB, Flyweight + DB schema // REST API + JSON

4.3.2 Deployment

The code is packaged by Maven and deployed as WAR file to AWS EB instance, running Tomcat 8 server. There is no need to configure anything with regards to basic networking - AWS EB is a back-to-back fully managed Platform as a Service (PaaS).

There are multiple things to consider when deploying to AWS EB, even though it seems "super-easy" in most instructional videos and ads:

1. Contrary to programmer's logic, one has to first create an "Application" and under that custom "Environments". In other words, Application is the main context, and environments are servers running in the same context, by default having the permission to communicate between each other.
2. In Environment setup, we can choose either a Web Server Environment or a Worker Environment. Web Server is the hub of any application and is used in both Semantic Data Manager and Tracking Engine. Workers are only used in Tracking Engine and will be explained later.
3. Chosen configuration was obviously Tomcat and I also opted for automatic load balancing and scaling.
4. Opting to automatically create an RDS instance along with the environment is a really bad design. Once an environment is terminated, so is the database instance which causes a serious data loss.
5. In order to access the servers via SSH, it is necessary to define a EC2 Security Group and assign it accordingly. Otherwise, all outside access is prohibited.
6. It is also crucial to wisely choose the instance type. I opted for m3.medium, because Java applciations by themselves are quite demanding and I didn't want to risk being on the edge when strange errors occur because of insufficient memory capacity.
7. Permissions and roles are absolutely crucial when it is desired to connect the Environment with other AWS services, such as object storage (S3 = Simple Storage Service) or a messaging queue (SQS = Simple Queue Service).

The size of the instance is recommended for any application running JVM. The configuration is Intel Xeon E5-2670 v2 (Ivy Bridge), 4GB SSD storage and 3.75GB RAM. Any additional memory is handled via S3.

// JIRA - pagination problems, service account

4.3.3 User Interface

The component Semantic Data Manager provides REST API to be consumed by any kind of client capable of HTTP requests. Because my specialization are mobile applications, I chose to implement a mobile application for iOS as an administrating user interface for this component.

4.3.3.1 Application flow

SCREENSHOTS

4.3.3.2 Technologies used

Moya, tracking engine

4.4 Tracking Engine

Because the tracking engine was developed after the Semantic Data Manager, I opted to use Java again.

The code is deployed in the same fashion as the Semantic Data Manager.

4.5 Tracked Device

TBD

4.6 Time Series Data

4.6.1 Numerous

Numerous has two components and uses monolithic git repository to keep everything together. It is designed to be a standalone micro service and can be used outside of the scope of this solution.

4.6.1.1 Back-end

For computational convenience in future development (= data modeling, model training etc.), Python was selected to be the language this component will be written in.

The code is packaged in Docker image and is deployed in two EC2 t2.medium instances - one for the API and one for the database. The main difference between T2 and M3 is the processor computational power - T2 has Dual Core Intel Xeon 3.3GHz with Turbo. Also it is a Burstable Performance Instance, meaning that it is provided with a baseline level of CPU performance with the ability to burst above. T2 instances are for workloads that don't use the full CPU often or consistently, but occasionally need to burst (computing statistical models for example in my case).

4.6.1.2 Front-end

Requirements for mobile client were only for iOS devices. Therefore, as no code portability was required, I opted to go native. For native development, Apple's language Swift is used (as of March 2016, in version 2.2). Swift is a general-purpose, multi-paradigm (both object oriented and functional), compiled programming language. It was first released to support iOS and Mac OS X, now supporting also tvOS (Apple TV 4th generation and newer) and watchOS (Apple Watch). Many more cases of use are coming, because Swift compiler has been open-sourced. It is gaining popularity among non-Apple developers mainly due to its safety, robustness and ease of use.

Dependencies

// CocoaPods

User Interface

DASHBOARD screenshot + DETAIL screenshot

Color scheme of the application was provided by a graphical designer. Most of the graphics is set on code level. I am not in favor of bloated projects because of multitude of png files for every possible device. Code generated graphics may introduce some level of complexity, but the space saved on user's device is more important. Even with app-slicing (method of distribution provided by Apple - only the resources needed for your device are downloaded), the amount of space saved is at least 3MB.

// ios-charts

Networking

// Alamofire + VPN

Data Model

// Realm.IO

Chapter 5

Testing

- Způsob, průběh a výsledky testování.
- Srovnání s existujícími řešeními, pokud jsou známy.

Chapter 6

Conclusion

6.1 Future development

The current solution can be deployed in testing environment as an MVP (= Minimum Viable Product). In order for it to run in production environment, legal steps have to be carried out in order to be fully compliant.

6.2 Technical debt

Currently unknown

6.3 Closing remarks

TBD

Bibliography

[1] M. Haindl, Ľ. Kment, and P. Slavík. Virtual information systems. In *WSCG'2000 — Short communication papers*, pages 22–27. University of West Bohemia, Pilsen, 2000.

[2] D. Radigan. Jql: the most flexible way to search jira. `http://blogs.atlassian.com/2013/01/jql-the-most-flexible-way-to-search-jira-14/`, 29. 1. 2013.

[3] P. Slavík. Grammars and rewriting systems as models for graphical user interfaces. *Cognitive Systems*, 4(3/4):381–399, 1997.

[4] Spring boot reference guide. `http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/`, up to date 16. 5. 2016.

[5] P. Webb. Spring boot – simplifying spring for everyone. `https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone`, 6. 8. 2013.

[6] CSTUG — $\mathcal{CS}$TEX Users Group — hlavní stránka. `http://www.cstug.cz/`, stav z 2. 3. 2009.

[7] K336 Info — pokyny pro psaní diplomových prací. `https://info336.felk.cvut.cz/clanek.php?id=400`, stav ze 4. 5. 2009.

[8] Knihovna Grafické skupiny. `http://www.cgg.cvut.cz/Bib/library/`, stav z 30. 8. 2001.

[9] Grafický vektorový editor pro práce vhodný pro práci LATEXem. `http://tclab.kaist.ac.kr/ipe/`, stav z 4. 5. 2009.

[10] LATEX — online manuál. `http://www.cstug.cz/latex/lm/frames.html`, stav ze 4. 5. 2009.

[11] Wiki books LATEX. `http://en.wikibooks.org/wiki/LaTeX/`, stav z 3. 4. 2009.

[12] J. Žára, B. Beneš, and P. Felkel. *Moderní počítačová grafika*. Computer Press s.r.o, Brno, 1st edition, 1998. In Czech.

Appendix A

List of abbreviations

API Application Programming Interface

AWS Amazon Web Services

CSRF/XSRF Cross-site Request Forgery

EB Elastic Beanstalk

EC2 Elastic Compute Cloud

JSON JavaScript Object Notation

KPI Key Performance Indicator

MVP Minimum Viable Product

PaaS Platform as a Service

PoC Proof of Concept

RDS Relational Database Service

S3 Simple Storage Service

SDLC Software Development Life Cycle

UX User Experience

Appendix B

Instalační a uživatelská příručka

Tato příloha velmi žádoucí zejména u softwarových implementačních prací.

Appendix C

Obsah přiloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce. (viz [7]):

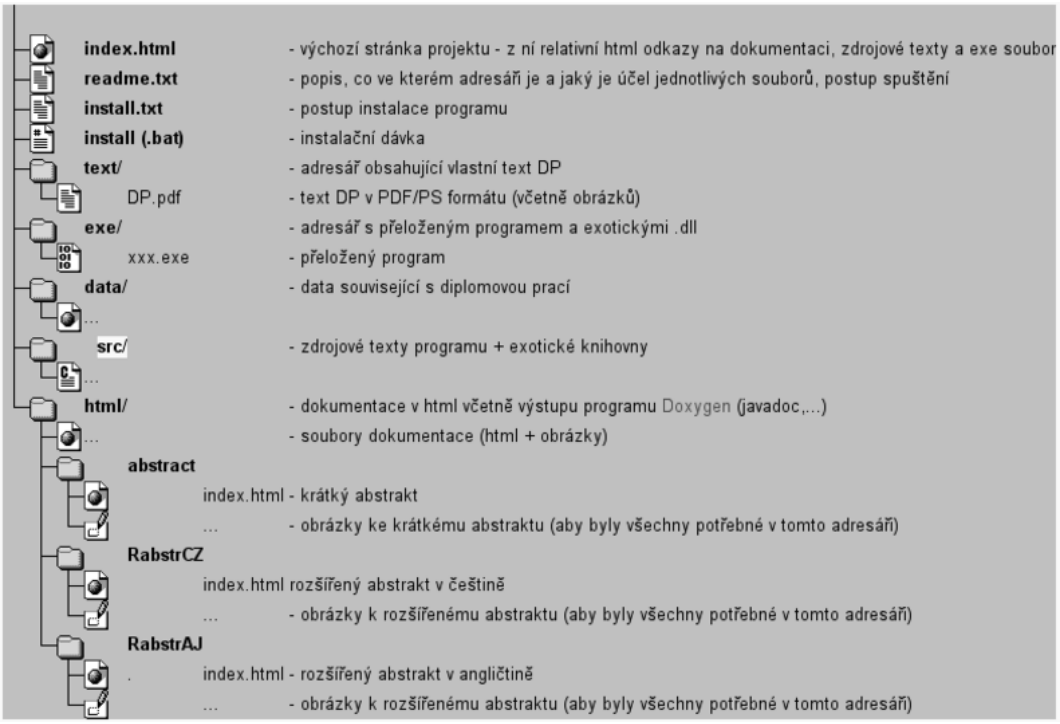


Figure C.1: Seznam přiloženého CD — příklad

Na GNU/Linuxu si strukturu přiloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případne index.html apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.

Appendix D

Pokyny a návody k formátování textu práce

Tato příloha samozřejmě nebude součástí vaší práce. Slouží pouze jako příklad formátování textu.

Používat se dají všechny příkazy systému LaTeX. Existuje velké množství volně přístupné dokumentace, tutoriálů, příruček a dalších materiálů v elektronické podobě. Výchozím bodem, kromě Googlu, může být stránka CSTUG (Czech Tech Users Group) [6]. Tam najdete odkazy na další materiály. Vetšinou dostačující a přehledně organizovanou elektronikou dokumentaci najdete například na [10] nebo [11].

Existují i různé nadstavby nad systémy TeX a LaTeX, které výrazně usnadní psaní textu zejména začátečníkům. Velmi rozšířený v Linuxovém prostředí je systém Kile.

D.1 Vkládání obrázků

Obrázky se umísťují do plovoucího prostředí `figure`. Každý obrázek by měl obsahovat **název** (`\caption`) a **návěští** (`\label`). Použití příkazu pro vložení obrázku `\includegraphics` je podmíněno aktivací (načtením) balíku graphicx příkazem `\usepackage{graphicx}`.

Budete-li zdrojový text zpracovávat pomocí programu `pdflatex`, očekávají se obrázky s příponou `*.pdf`[1], použijete-li k formátování `latex`, očekávají se obrázky s příponou `*.eps`.[2]

Příklad vložení obrázku:

```
\begin{figure}[h]
\begin{center}
\includegraphics[width=5cm]{figures/LogoCVUT}
\caption{Popiska obrazku}
\label{fig:logo}
```

[1] pdflatex umí také formáty PNG a JPG.

[2] Vzájemnou konverzi mezi snad všemi typy obrazku včetně změn vekostí a dalších vymožeností vám může zajistit balík ImageMagic (http://www.imagemagick.org/script/index.php). Je dostupný pod Linuxem, Mac OS i MS Windows. Důležité jsou zejména příkazy convert a identify.

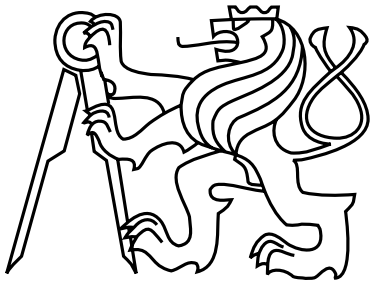


Figure D.1: Popiska obrázku

DTD	construction	elimination
|	`in1|A|B a:sum A B` `in1|A|B b:sum A B`	`case([_:A]a)([_:B]a)ab:A` `case([_:A]b)([_:B]b)ba:B`
+	`do_reg:A -> reg A`	`undo_reg:reg A -> A`
∗, ?	the same like | and + with `emtpy_el:empty`	the same like | and + with `emtpy_el:empty`
R(a,b)	`make_R:A->B->R`	`a: R -> A` `b: R -> B`

Table D.1: Ukázka tabulky

```
\end{center}
\end{figure}
```

D.2 Kreslení obrázků

Zřejmě každý z vás má nějaký oblíbený nástroj pro tvorbu obrázků. Jde jen o to, abyste dokázali obrázek uložit v požadovaném formátu nebo jej do něj konvertovat (viz předchozí kapitola). Je zřejmě vhodné kreslit obrázky vektorově. Celkem oblíbený, na ovládání celkem jednoduchý a přitom dostatečně mocný je například program Inkscape.

Zde stojí za to upozornit na kreslící programe Ipe [9], který dokáže do obrázku vkládat komentáře přímo v latexovském formátu (vzroce, stejné fonty atd.). Podobné věci umí na Linuxové platformě nástroj Xfig.

Za pozornost ještě stojí schopnost editoru Ipe importovat obrázek (jpg nebo bitmap) a krelit do něj latexovské popisky a komentáře. Výsledek pak umí exportovat přímo do pdf.

D.3 Tabulky

Existuje více způsobů, jak sázet tabulky. Například je možno použít prostředí `table`, které je velmi podobné prostředí `figure`.

Zdrojový text tabulky D.1 vypadá takto:

```
\begin{table}
\begin{center}
\begin{tabular}{|c|l|l|}
\hline
\textbf{DTD} & \textbf{construction} & \textbf{elimination} \\
\hline
$\mid$ & \verb+in1|A|B a:sum A B+ & \verb+case([_:A]a)([_:B]a)ab:A+\\
&\verb+in1|A|B b:sum A B+ & \verb+case([_:A]b)([_:B]b)ba:B+\\
\hline
$+$&\verb+do_reg:A -> reg A+&\verb+undo_reg:reg A -> A+\\
\hline
$*,?$& the same like $\mid$ and $+$ & the same like $\mid$ and $+$\\
& with \verb+emtpy_el:empty+ & with \verb+emtpy_el:empty+\\
\hline
R(a,b) & \verb+make_R:A->B->R+ & \verb+a: R -> A+\\
 & & \verb+b: R -> B+\\
\hline
\end{tabular}
\end{center}
\caption{Ukázka tabulky}
\label{tab:tab1}
\end{table}
\begin{table}
```

D.4 Odkazy v textu

D.4.1 Odkazy na literaturu

Jsou realizovány příkazem `\cite{odkaz}`.

Seznam literatury je dobré zapsat do samostatného souboru a ten pak zpracovat programem bibtex (viz soubor `reference.bib`). Zdrojový soubor pro `bibtex` vypadá například takto:

```
@Article{Chen01,
  author  = "Yong-Sheng Chen and Yi-Ping Hung and Chiou-Shann Fuh",
  title   = "Fast Block Matching Algorithm Based on
             the Winner-Update Strategy",
  journal = "IEEE Transactions On Image Processing",
  pages   = "1212--1222",
  volume  =  10,
  number  =   8,
  year    = 2001,
}

@Misc{latexdocweb,
```

```
  author  = "",
  title   = "{\LaTeX} --- online manuál",
  note    = "\verb|http://www.cstug.cz/latex/lm/frames.html|",
  year    = "",
}
...
```

Pozor: Sazba názvů odkazů je dána BibTEX stylem (`\bibliographystyle{abbrv}`). BibTEX tedy obvykle vysází velké pouze počáteční písmeno z názvu zdroje, ostatní písmena zůstanou malá bez ohledu na to, jak je napíšete. Přesněji řečeno, styl může zvolit pro každý typ publikace jiné konverze. Pro časopisecké články třeba výše uvedené, jiné pro monografie (u nich často bývá naopak velikost písmen zachována).

Pokud chcete BibTEXu napovědět, která písmena nechat bez konverzí (viz `title = "{\LaTeX} --- online manuál"` v předchozím příkladu), je nutné příslušné písmeno (zde celé makro) uzavřít do složených závorek. Pro přehlednost je proto vhodné celé parametry uzavírat do uvozovek (`author = "..."`), nikoliv do složených závorek.

Odkazy na literaturu ve zdrojovém textu se pak zapisují:

```
Podívejte se na \cite{Chen01},
další detaily najdete na \cite{latexdocweb}
```

Vazbu mezi soubory `*.tex` a `*.bib` zajistíte příkazem `\bibliography{}` v souboru `*.tex`. V našem případě tedy zdrojový dokument `thesis.tex` obsahuje příkaz `\bibliography{reference}`.

Zpracování zdrojového textu s odkazy se provede postupným voláním programů `pdflatex <soubor>` (případně `latex <soubor>`), `bibtex <soubor>` a opět `pdflatex <soubor>`.[3]

Níže uvedený příklad je převzat z dříve existujících pokynů studentům, kteří dělají svou diplomovou nebo bakalářskou práci v Grafické skupině.[4] Zde se praví:

```
...
j) Seznam literatury a dalších použitých pramenů, odkazy na WWW stránky, ...
 Pozor na to, že na veškeré uvedené prameny se musíte v textu práce
 odkazovat -- [1].
Pramen, na který neodkazujete, vypadá, že jste ho vlastně nepotřebovali
a je uveden jen do počtu. Příklad citace knihy [1], článku v časopise [2],
stati ve sborníku [3] a html odkazu [4]:
[1] J. Žára, B. Beneš;, and P. Felkel.
     Moderní počítačová grafika. Computer Press s.r.o, Brno, 1 edition, 1998.
     (in Czech).
```

[3] První volání `pdflatex` vytvoří soubor s koncovkou `*.aux`, který je vstupem pro program `bibtex`, pak je potřeba znovu zavolat program `pdflatex` (`latex`), který tentokrát zpracuje soubory s příponami `.aux` a `.tex`. Informaci o případných nevyřešených odkazech (cross-reference) vidíte přímo při zpracovávání zdrojového souboru příkazem `pdflatex`. Program `pdflatex` (`latex`) lze volat vícekrát, pokud stále vidíte nevyřešené závislosti.

[4] Několikrát jsem byl upozorněn, že web s těmito pokyny byl zrušen, proto jej zde přímo necituji. Nicméně příklad sám o sobě dokumentuje obecně přijímaný konsensus ohledně citací v bakalářských a diplomových pracích na KP.

```
[2] P. Slavík. Grammars and Rewriting Systems as Models for Graphical User
    Interfaces. Cognitive Systems, 4(4--3):381--399, 1997.
[3] M. Haindl, Š. Kment, and P. Slavík. Virtual Information Systems.
    In WSCG'2000 -- Short communication papers, pages 22--27, Pilsen, 2000.
    University of West Bohemia.
[4] Knihovna grafické skupiny katedry počítačů:
    http://www.cgg.cvut.cz/Bib/library/
```

... abychom výše citované odkazy skutečně našli v (automaticky generovaném) seznamu literatury tohoto textu, musíme je nyní alespoň jednou citovat: Kniha [12], článek v časopisu [3], příspěvek na konferenci [1], www odkaz [8].

D.4.2 Odkazy na obrázky, tabulky a kapitoly

- Označení místa v textu, na které chcete později čtenáře práce odkázat, se provede příkazem `\label{navesti}`. Lze použít v prostředích `figure` a `table`, ale též za názvem kapitoly nebo podkapitoly.
- Na návěští se odkážeme příkazem `\ref{navesti}` nebo `\pageref{navesti}`.

D.5 Rovnice, centrovaná, číslovaná matematika

Jednoduchý matematický výraz zapsaný přímo do textu se vysází pomocí prostředí `math`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$`.

Kód `$ S = \pi * r^2 $` bude vysázen takto: $S = \pi * r^2$.

Pokud chcete nečíslované rovnice, ale umístěné centrovaně na samostatné řádky, pak lze použít prostředí `displaymath`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$$`. Zdrojový kód: `|$$ S = \pi * r^2 $$|` bude pak vysázen takto:

$$S = \pi * r^2$$

Chcete-li mít rovnice číslované, je třeba použít prostředí `eqation`. Kód:

```
\begin{equation}
  S = \pi * r^2
\end{equation}

\begin{equation}
  V = \pi * r^3
\end{equation}
```

je potom vysázen takto:

$$S = \pi * r^2 \tag{D.1}$$

$$V = \pi * r^3 \tag{D.2}$$

D.6 Kódy programu

Chceme-li vysázet například část zdrojového kódu programu (bez formátování), hodí se prostředí `verbatim`:

```
         (* nickname2 *)
Lego> Refine in1
             (do_reg (nickname1 h));
Refine by  in1 (do_reg (nickname1 h))
   ?4 : pcdata
   ?5 : pcdata
          (* surname2 *)
Lego> Refine surname1 h;
Refine by  surname1 h
   ?5 : pcdata
          (* email2 *)
Lego> Refine undo_reg (email1 h);
Refine by  undo_reg (email1 h)
*** QED ***
```

D.7 Další poznámky

D.7.1 České uvozovky

V souboru `k336_thesis_macros.tex` je příkaz `\uv{}` pro sázení českých uvozovek. „Text uzavřený do českých uvozovek.“