

Nikolai Denissov

Prototype for Research Infrastructure Databank Service

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

8.5.2018

Author(s)	Nikolai Denissov
Title	Prototype for Research Infrastructure Databank Service
Number of Pages	38 pages
Date	Tuesday 8 th May, 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Patrick Ausderau, Senior Lecturer Stina Westman, D.Sc. (Tech.), Director
<p>Data integration is a common process for projects aiming at reporting. National Research Information Hub is an example of data integration projects, which aim to provide insight and uncover interdependencies in the education field in Finland. A part of this project - Research Infrastructure (RI) Databank Service - focuses on harvesting data related to research infrastructures.</p> <p>Recent changes in RI Databank Service scope introduced organizational and architectural challenges implying automated integrations with partner universities' systems. This thesis aims to address new requirements by creating a prototype for the next improved version of RI Databank Service application. The goal of the prototype implementation is to evaluate architectural approach viability and to verify the technological compatibility of the software components. The thesis is a report of lessons learnt during the service development.</p> <p>The results show that the application created in this project is able to integrate with common data format used in universities' systems. Also, it is capable of adjusting to the changing data model. Performance testing has confirmed that data transformation speed satisfies the requirements. In contrast with the previous service version, the prototype is simpler, despite the overall increase in the functionality. Usage of code generation tools accelerated the development process. Built prototype will be used as a base for further RI Databank Service project development at CSC.</p>	
Keywords	Code generation, Data Integration, JPA, Mapping, XSLT, XML parsing

Contents

1	Introduction	1
2	Current State Analysis	3
2.1	Research Infrastructure Databank Service	3
2.1.1	General description	3
2.1.2	Technical description	4
2.2	Data Management and Partner Systems	7
3	Theoretical background	10
3.1	Integration	10
3.2	Data formats and processing	11
3.3	Mapping	14
3.4	Code generation	14
4	Methods and materials	16
4.1	Methods	16
4.2	Requirements and development environment	17
5	Solution	20
5.1	Data processing	22
5.2	Data models	23
5.3	Mapping	24
5.4	REST API	25
5.5	Testing	26
6	Discussion	28
7	Conclusion	32
	Bibliography	33

Abbreviations

API	Application Programming Interface.
CERIF	Common European Research Information Format.
CI/CD	Continuous Integration and Continuous Deployment.
CRIS	Current Research Information System.
CSC	Finnish IT Center for Science Ltd.
DB	Database.
DOM	Document Object Model.
DTD	Document Type Definition.
ETL	Extract, Transform, Load.
HTML	HyperText Markup Language.
HTTP	HyperText Transfer Protocol.
IDE	Integrated Development Environment.
IOW	InterOperability Workbench.
IPUMS	Integrated Public Use Microdata Series.
JAXB	Java Architecture for eXtensible Markup Language (XML) Binding.
JDBC	Java Database Connectivity.
JDK	Java Development Kit.
JPA	Java Persistence Application Programming Interface (API).
JRE	Java Runtime Environment.
JSON	JavaScript Object Notation.
OOP	Object-Oriented Programming.
ORM	Object-Relational Mapping.
PID	Persistent Identifier.
PoC	Proof of Concept.
POJO	Plain Old Java Object.
REST	Representational State Transfer.
RI	Research Infrastructure.
SAX	Simple API for XML.
SOAP	Simple Object Access Protocol.
SQL	Structured Query Language.
StAX	Streaming API for XML.
TIPA	Research Infrastructure Databank in Finnish: <i>"Tutkimusinfrastruktuurien tietopankki"</i> .
TrAX	Transformation API for XML.
UI	User Interface.
UML	Unified Modelling Language.
XML	eXtensible Markup Language.
XSD	XML Schema Definition.
XSL	eXtensible Stylesheet Language.
XSLT	eXtensible Stylesheet Language Transformations.

Glossary

.JAR	(Java Archive) file format that enables to bundle multiple files into a single archive file. Typically a JAR file contains the class files and auxiliary resources associated with applets and applications [1].
.WAR	(Web application Archive) files used to distribute Java-based Web applications. A .WAR has the same file structure as a .JAR file, which is a single compressed file that contains multiple files bundled inside it [2].
GitHub	git-based source code repository service.
Java	is a general-purpose, concurrent, strongly typed, class-based object-oriented language [3].
JavaScript	high-level, interpreted programming language.
Leaflet	open-source JavaScript library for mobile-friendly interactive maps [4].
Liferay	Java-based Digital Experience Platform.
Linked Data	a paradigm for publishing data on the Web and collaborating with users and machines [5].
Maven	build automation tool for Java projects.
portlet	application used by a portal website to receive requests from clients and return information [6].
Service Builder	model-driven code generation tool built by Liferay [7].
Spring	Java platform that provides comprehensive infrastructure support for developing Java applications [8].
Spring Boot	Spring solution based on convention over configuration principle for building production-ready Spring applications [9].
Swagger	framework of API developer tools for the OpenAPI Specification (OAS).
TEKES	Finnish Funding Agency for Innovation.
Vaadin	Java User Interface (UI) framework for building single page Web applications.
Web Service	software service or application used to communicate between devices on a network in a standardized way [10].

1 Introduction

Data integration is a common process for projects aiming at reporting. However, collection, storage and modification of data from several sources might be performed in various ways. A good example of such data integration projects is the National Research Information Hub project, which aims at providing insight and uncovering interdependencies in the education field in Finland. A part of this project - Research Infrastructure (RI) Databank Service - focuses on harvesting data related to research infrastructures. The service is developed at Finnish IT Center for Science Ltd (CSC).

CSC is a special task company in Finland. It is dedicated to support and provide expert ICT services for research, education, culture, public administration and enterprises. Company employs approximately 320 experts in Espoo and Kajaani. Apart from doing development project work for the Ministry of Culture and Education, universities and research institutes the company is known for connecting Finland to the Internet and its supercomputing capacity offer. [11]

Recent changes in RI Databank Service scope introduced organizational and technical (architectural) challenges. For example, instead of manual data collection, automated integrations with partner universities' systems must be established. This thesis aims to address new requirements by creating a prototype for the next improved version of RI Databank Service application. Subtasks for the work are verifying selected technologies' compatibility and exposing the issues faced during the implementation. Also, the viability of the overall approach is tested.

The thesis is divided into two larger parts. In the first part, the current state of RI Databank Service is described from both organizational and technical angles. In addition, example-based description of integration partners' Current Research Information System (CRIS) is provided. Moreover, theoretical background is defined by introducing key relevant concepts and tools suitable for the prototype development. The latter part of the work is focused on the practical implementation aspects. The methodological approach as well as development environment are described and details of the created solution

are presented. Finally, the project results are reported and discussed in the light of the theoretical assumptions.

2 Current State Analysis

2.1 Research Infrastructure Databank Service

2.1.1 General description

Research Infrastructure Databank Service is a service developed as part of the Open Science and Research Initiative 2014-2017 funded by the Ministry of Education and Culture in Finland [12]. As the name indicates, the initiative aimed at promoting research information availability and open science. This was done through developing several collaboration approaches and provision of services. The key stakeholders of the initiative were researchers, research organizations, Finnish institutions of higher education, funding bodies including Academy of Finland and TEKES as well as the National Library of Finland [13]. As the service description documentation states, RIs are

”research facilities, equipment, materials and services that enable research and development at various stages of innovation, while supporting organized research, research training and teaching, and developing research and innovation capacity. [14]”

Research Infrastructure Databank Service became available in early 2016 as part of AVAA publishing platform¹ [13]. Until the end of 2017 the service was operating in a piloting phase. The primary goal of the phase was to aggregate and provide open access to the data related to the research infrastructures Finland owns or participates in. Potential customers for the service are researchers, research infrastructure service providers and funders [13]. Piloting phase aimed at finding out the existence of the required data at the universities and other RI-handling organizations. Currently, the service contains data on RIs, which are included into Finland’s Strategy and Roadmap for Research Infrastructures 2014-2020 [16]. Also, the service addressed an absence of common Persistent Identifiers (PIDs) for Finnish RIs. The identifier solution the service has provided is a generation of unique PIDs inside the application. The identifiers are exposed via RI Data-

¹research information publishing platform offered by the Ministry of Education and Culture [15]

bank Application Programming Interface (API) and National Library of Finland identifier resolution service [17]. The resulting registry owner is Academy of Finland. Information management and component development are done by the service provider - CSC [13].

Since the end of the Open Science and Research Initiative, from the beginning of 2018, the RI Databank Service was structurally moved under a larger development context - the National Research Information Hub. The updated goal for the project is to act as a data provider for the holistic access to research information in the field of science and education. The source data and metadata are aggregated from different registers and systems [18] [19]. In this new context, RI Databank will operate as data supplier and system integrator. On the basis of the piloting phase results, project will be redeveloped to establish real integrations with partners' systems.

2.1.2 Technical description

RI Databank Service was initially implemented as a part of AVAA, technically a Liferay portal application depending on diverse Database (DB) types. RI Databank Service is one of the portlets in the AVAA application-rich environment. Application uses Java as the main programming language. MariaDB database is used to preserve the data. Liferay's Service Builder framework is employed to build persistence and API layers. To create a view layer, Vaadin open-source framework is used along with some extensions (e.g. Leaflet). Apache Maven is responsible for building and packaging the application. The ready made .WAR package is delivered to the platform via Liferay's default deployment mechanisms. In a wider context, RI Databank Service is a Java Rich Internet Application with some usage of spatial data [20].

In compliance with the original initiative's² nature, open-source tools and technologies were intensively utilized in the service development. Just as all the other application codebases for AVAA platform, RI Databank Service's source code is licensed under GNU Affero General Public License terms and available on GitHub³ [21] by the name of Research Infrastructure Databank in Finnish: "*Tutkimusinfrastruktuurien tietopankki*" (TIPA).

²Open Science and Research Initiative 2014-2017

³available at <https://github.com/avaa-csc/avaa-tipa/>

TIPA as on end of 2017

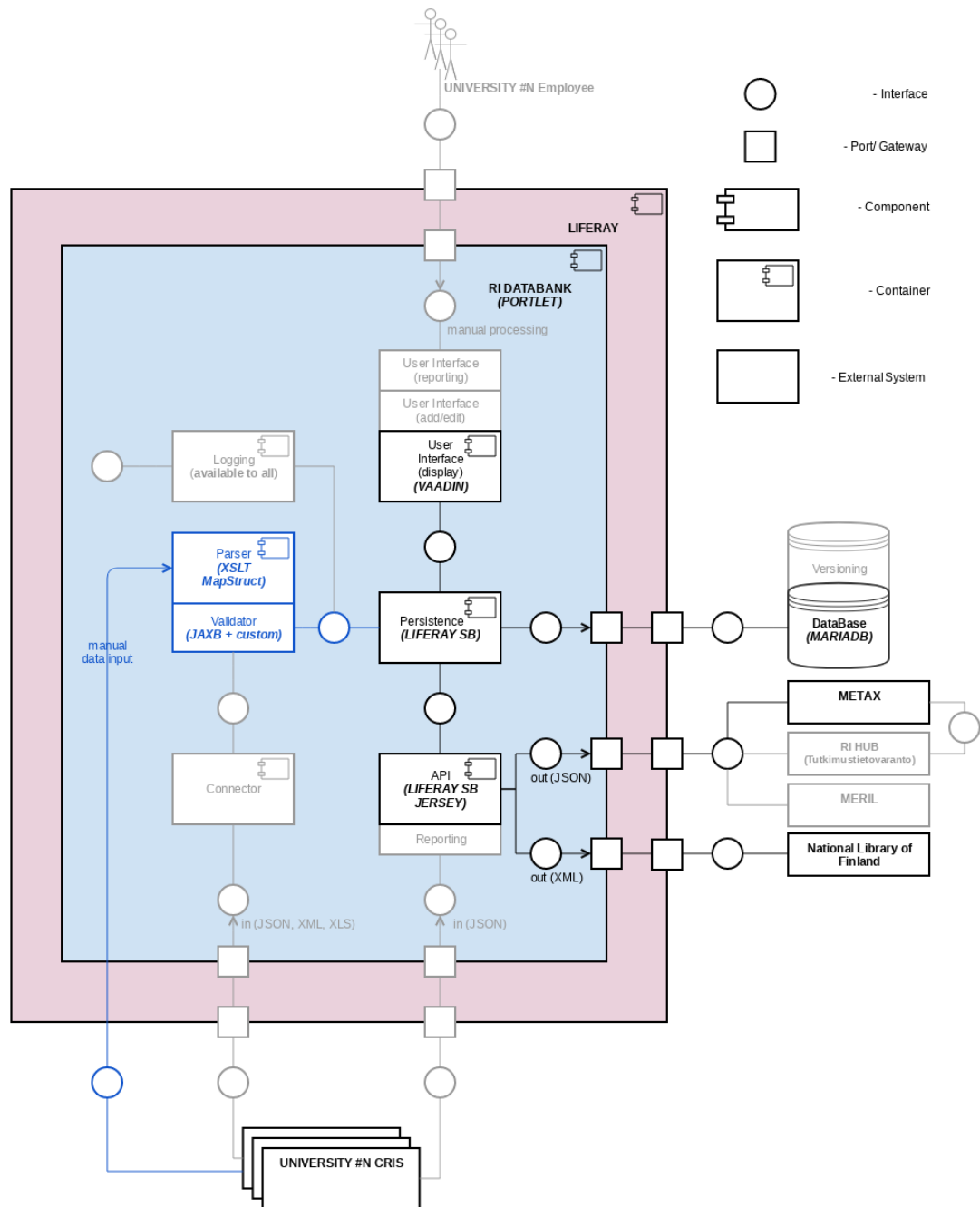


Figure 1: Research Infrastructure Databank Service component schema. Extracted from CSC's internal documentation (2017) [22]

Diagram⁴ depicted in figure 1 represents the application structure layers. Borders are painted black for the RI Databank Service functional parts existing by the end of 2017, and components with dark blue borders represent the parts to be developed in the next project

⁴component diagram uses elements of the standard Unified Modelling Language (UML) notation

phase. The schema depicted in the figure 1 also demonstrates that service contains a set of APIs opened for public consumption. API marked as "out (JavaScript Object Notation (JSON))" - is a rich API derived from Liferay's Service Builder functionality [23]. API referred as "out (eXtensible Markup Language (XML))" is a simple Representational State Transfer (REST) API [24] providing an XML output. The endpoint exposes the link between human-readable RI description Web page and its generated PIDs.

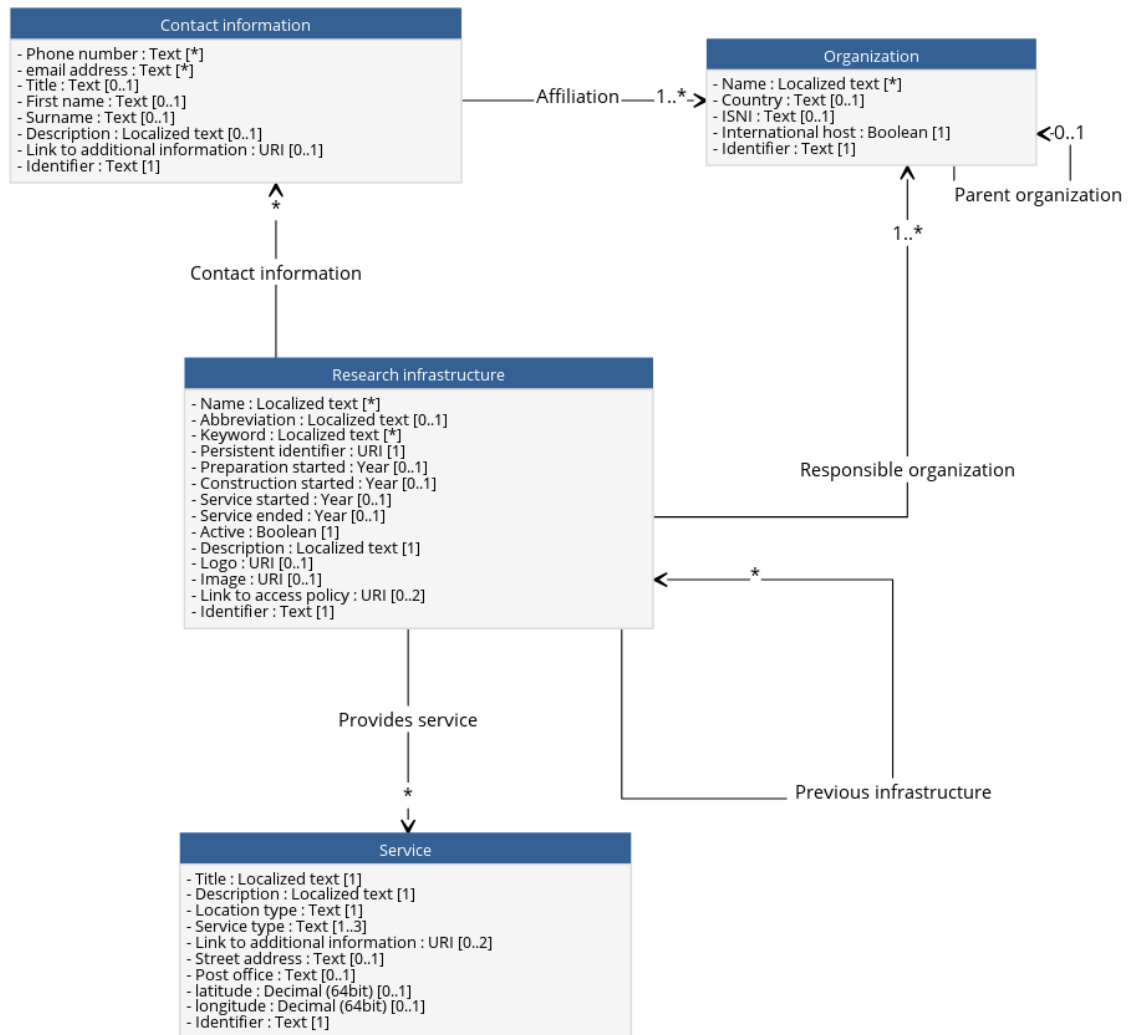


Figure 2: Research Infrastructure Databank Service simplified data model. Copied from CSC's InterOperability Workbench (IOW) tool user interface (2017) [25]

RI Databank Service simplified data model for the existing service in figure 2 shows the basic conceptual entities and their interrelations. Model depicts relationships between Research Infrastructures, Organizations and Services provided. Research Infrastructure has a key role in the model. It contains data on PIDs, current and former names of the Infrastructure, its type and references to the descriptive Web pages. The data model

along with semantic references are stored in a separate service called IOW. The service is publicly available for the interested parties, enhancing interoperability in exchanging research information within the domain and beyond. [25]

During the piloting stage, the actual application data was collected manually via Excel spreadsheets. Collected data was converted into Structured Query Language (SQL) insert statements and imported into the DB storage. Data also required manual purification. Direct parsing of Excel spreadsheets was not possible due to poor data integrity. The approach rendered to be slow, error-prone and highly labour-intensive.

Manual data collection and processing phase of the project has confirmed the existence of the target data for the service, but also brought up the issues associated with the source data gathering approach. To address these issues and keep the information up to date, establishing integrations with universities' RI Management Systems was planned as a next service development stage.

2.2 Data Management and Partner Systems

While preparing project continuation roadmap for 2018, a survey amidst Finnish universities was held. In the questionnaire, universities were asked to describe what CRIS they are using, and give an estimate on the amount of the RIs related to their organization. Results, produced by the survey are displayed in figure 3.

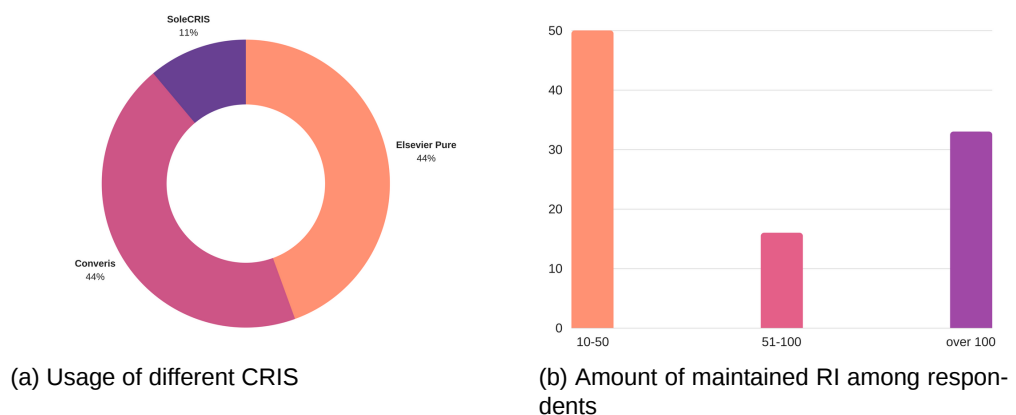


Figure 3: Results of the survey *"Suomen tutkimusinfrastruktuurit - palvelun tiedonsiirron pilotointi"*, conducted among the Finnish universities. Visualized based on Kainulainen (2017) [26]

All of nine universities, which have taken part in the survey are either using or in the process of implementing the CRIS solution [26]. Typically, CRIS systems are used to record and manage research information on RIs or other entities such as publications, projects etc., and can thus be used as a data source for RI Databank Service next development stage. University of Helsinki actively contributed during the pilot stage of the project and agreed to become the first partner for integration in the next project stage. The university is using Elsevier Pure - a widely spread general-purpose CRIS in Finland [26].

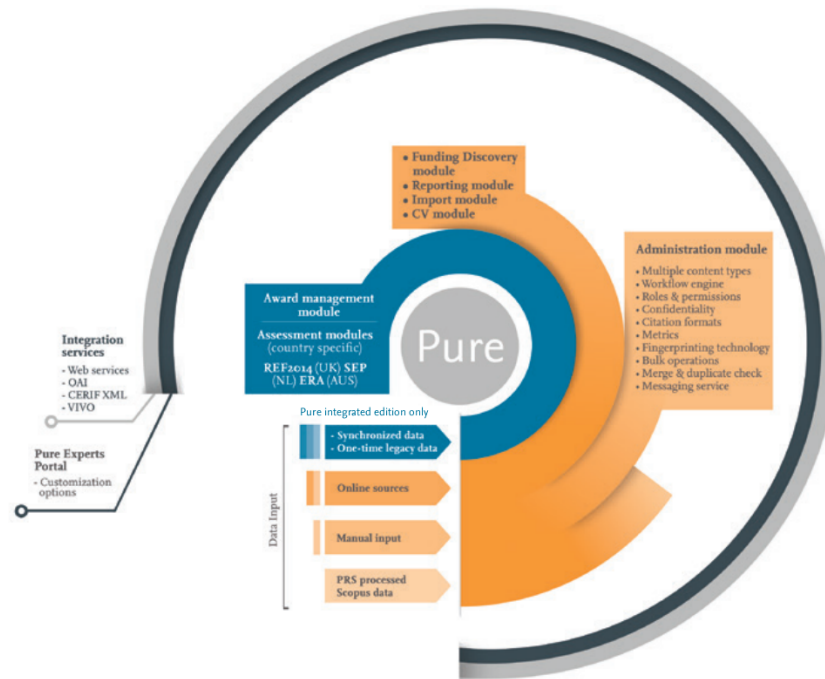


Figure 4: Elsevier Pure conceptual structure. Copied from "Pure hosted edition" brochure. Elsevier (2014) [27]

Elsevier Pure is a research intelligence solution that helps organizing and managing data as well as processes in the field of scientific research [28]. The solution has a modular structure, which allows to support on-demand National Assessments for multiple countries (UK, Netherlands, Australia) [29]. Key application modules are: Capture, Validate, Profile, Identify, Report, Analyze, Monitor and Showcase [30].

Different point of view on modularizing is presented in the figure 4, which also shows presence of the system integration services module. This module supports diverse data export formats as a standalone part of the system, which is a natural component for modern CRIS solutions. Pure supports Common European Research Information Format (CERIF) XML as well as more traditional Web Services approach. API calls provide results in two possi-

ble formats: XML and JSON. To ease the integration process, Elsevier Pure APIs are well documented. According to the latest API description trends there is a Swagger-like documentation available for the REST API and a set of definitions for Simple Object Access Protocol (SOAP) endpoints [31].

3 Theoretical background

3.1 Integration

As a term, integration in general, and a subterm Data Integration, have very wide definitions. For example IBM suggests [32] the following definition: "Data integration is the combination of technical and business processes used to combine data from disparate sources into meaningful and valuable information". Simple definition of integration, applicable for the software development context, describes it as a process of seamless provision of end user with an aggregated data from various sources [33] [34]. Further references to the term in the work use this simple definition of integration.

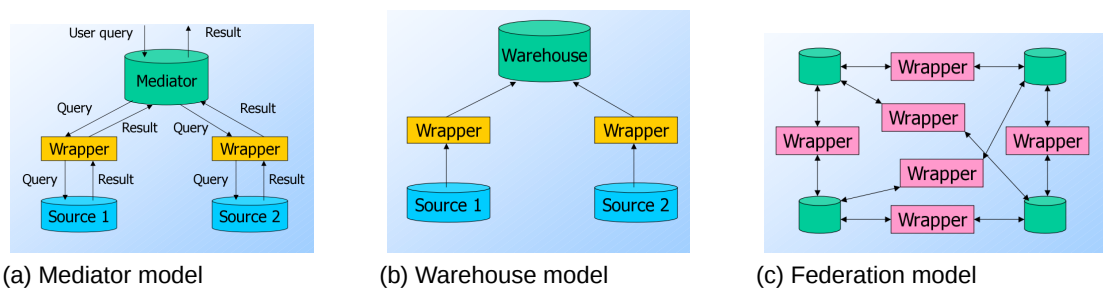


Figure 5: Visualizations of Data integration models. Extracted from Ullman (2007) [35]

The concept of integration started to evolve since the early stages of distributed service development. One of the first and influential projects in the field was Integrated Public Use Microdata Series (IPUMS), where the data warehousing approach was introduced [36]. Later on, with the development of the Web, other major architectural approaches were created: mediation and federation systems [37].

The main integration architectures are as follows [35]:

- Federation: each party directly communicates with others.
- (Data) Warehouse: original data is translated into higher level schema (often called a global schema) and copied into common storage; querying of the data is direct.
- Mediator: similar to the Warehouse model, but the data persists at original sources and is prompted via artificial gateway; corresponding parts of the query are redi-

rected to original sources and responses are aggregated.

The noteworthy aspects of data integration models are the physical location of the data, required mapping efforts and the resulting querying speed. In the context of mapping complexity, it is important to observe that models with central orchestrator are considered to be more effective than the distributed ones. The role of the orchestrator is reducing complexity⁵ from $O(N^2)$ to $O(N)$ [39, p. 9]. The visual representation of non-orchestrated and orchestrated model types is shown in the figure 6.

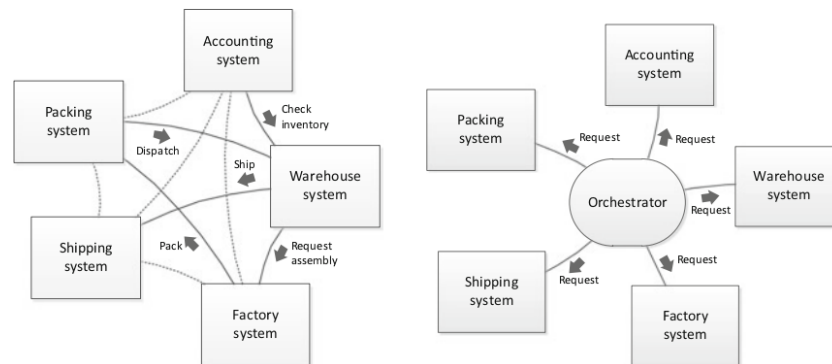


Figure 6: Non-orchestrated and orchestrated model types for data integration. Adapted from Ferreira (2011) [39, p. 9]

Additional dimension of the integration process is related to the nature of the data source. Historically used unstructured data formats (files, databases query outputs, etc.) were challenging from integration point of view as the data was not self-descriptive [40, pp. 2-3]. Introduction of the first XML standard provoked rapid development in the industry as it simplified data processing. Recent trends in the field are focused on unifying structured data descriptions, e.g. advanced concepts such as Linked Data and Semantic Web⁶. Such developments promote data auto-integration.

3.2 Data formats and processing

After several decades of the active usage, XML still remains a widely spread technology. Even today, XML is being used as an internal (canonical)⁷ data model representation.

⁵Big O notation is used in Computer Science to describe the performance or complexity of an algorithm [38]

⁶a proposed development of the World Wide Web in which data in Web pages is structured and tagged in such a way that it can be read directly by computers [41]

⁷a new data handling format created solely for the integration purposes [39, pp. 105-106]

One of the XML definitions characterizes it as a self-descriptive markup language for the data storage and transport [42]. The technology originates from the field of a large-scale electronic publishing, subsequently achieving greater role in data interchange on the Web and elsewhere. XML has become a *de-facto* standard format for the data transfer [43] [44, p.291].

Benefits of using XML format are: ease of extensibility, standardization and compliance with the Document Type Definition (DTD) or XML Schema Definition (XSD), which in practice significantly simplifies data validation [45]. Additionally XML processing (parsing) has become a trivial task with plenty of decomposition approaches available. There are multiple implementations for different approaches, for example: Simple API for XML (SAX), Streaming API for XML (StAX), Document Object Model (DOM), Transformation API for XML (TrAX) [46].

Certain reasonable concerns exist when comparing the performance of XML to other common data transfer formats, for example JSON. Several sources confirm, that JSON overperforms XML both in terms of processing speed and required resource usage [47] [48]. JSON, being a product of JavaScript environment, is easier to use with JavaScript applications [47]. On the other hand, XML as a data format has its own significant strengths like improved security, great validation features as well as simplicity of debugging and troubleshooting [48].

XML has a time-proven, feature saturated support among different programming languages. A practical example of such support in Java environment is the Java Architecture for XML Binding (JAXB). In contrast with SAX, StAX, DOM and TrAX parsers, JAXB provides more convenient approach for processing XML by directly mapping the contents of the document onto Java classes. The effort of the manual selecting and matching the fields is thus significantly reduced. This process, however, assumes the existence of a valid XSD [49]. Based on the schema, JAXB allows to complete several primary operations: marshalling⁸, unmarshalling⁹ and validation [51, p. 2]. Typical internal workflow for XML to code mapping in JAXB is described in figure 7.

⁸process of gathering data and transforming it into a standard format before it is transmitted over a network so that the data can transcend network boundaries [50]

⁹process opposite to marshalling

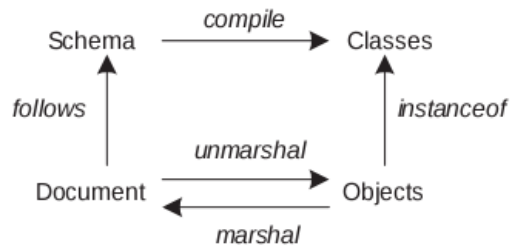


Figure 7: A mapping of XML to Java classes/objects. Copied from JAXB Specification, Fialli (2003) [51]

After the introduction of JSON, there is no single dominating standard for data interchange format any more. As follows, integration tools and technologies should support transformation of data in both formats. One of the options for data transformation in both formats is eXtensible Stylesheet Language Transformations (XSLT) specification, where eXtensible Stylesheet Language (XSL) is a member of the XML environment [52]. XSLT was initially defined as "a language for transforming XML documents into other XML documents, text documents or HyperText Markup Language (HTML) documents" [53].

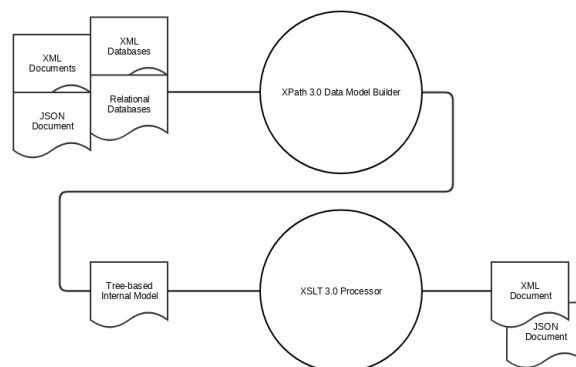


Figure 8: Processing with XSLT. Adaptation from W3C [53]

Since the specification version 3.0, XSLT is capable of processing additional formats including JSON [54]. Transformation language is suitable for cases with changing requirements for input and output data formats. Besides XSLT usage introduces additional abstraction layer into application architecture by decoupling inputs from internal data model [55, p. 3]. Schematic process of data processing with XSLT is shown in the figure 8.

3.3 Mapping

Implementation of objects mapping is often required during the software development process, provided that different data models have to coexist inside the application. In context of Object-Oriented Programming (OOP) languages, object mapping is defined as matching data sets or their parts to each other [56] [57].

In case of Java applications, the naive approach is a manual creation of direct mapping between classes' fields. This, however, results in introducing significant amount of hard-to-maintain verbose code. Several open source libraries have addressed this issue by designing object mapping tools. Based on the functioning principle, these tools can be conditionally divided into two large groups: reflection¹⁰ based mappers and the ones, where the mapping code is generated on pre-compilation phase. Approaches differ in speed and flexibility. Despite some advanced features like the ability to access private fields, recursion (reflection) based libraries show significantly lower performance results [60].

Modern mapping tools differ to the extent of community and Integrated Development Environment (IDE) support as well as documentation level. An example of recent developments in the field is MapStruct library. It operates by processing annotations responsible for mapping layer generation. Additionally, library is able to reduce the amount of mapping-related runtime errors. This happens because code generation is moved to pre-compilation phase and error-checking is therefore performed by compiler before the runtime [61] [62].

3.4 Code generation

Code generation process can be hardly universally defined. One of the most generic definitions explains it as a compiler's internal process of translating semantic language into "machine" language (bytecode) [63] [64]. A different approach to code generation originates from internal processes of the frameworks, where functionality of generated code layer is hidden behind an API. Descriptive examples of such frameworks include

¹⁰Java language feature to scrutinize and modify runtime objects [58] [59]

Spring¹¹ and Vaadin¹². The compiler's and frameworks' code generation procedure are similar in a way that once created, the generated code is either invisible or unreachable for the developer.

Some frameworks - e.g. Liferay's Service Builder - allow manual modification of the generated code. Based on object-relational mapping technology, it creates a persistence layer, also allowing to access the entities through an API. Additionally, generated code is in strict compliance with a separation of concerns principle [65]. Previously discussed JAXB tool also provides possibility of code generation with any further modification. JAXB can be used for generation of Java classes (objects) from a well-formed XSD schema [49]. Generated classes act as accommodation objects for mapped XML data and can be adjusted if needed.

The task of accessing and handling data stored in persistent storage (database) is to some extent present in most of the modern applications and can be considered trivial. To support this operation in Java environment Java Persistence API (JPA) was developed. Framework contains a collection of standard interfaces which allow to represent database information in a form of Plain Old Java Objects (POJOs) [66] [67]. These plain annotated Java entities are generated from the database schema based on the selected implementation. JPA, however, adds an extra abstraction layer, that allows to use common API regardless of the underlying implementation [68].

In general, code generation tools simplify software development process which is beneficial for all the parties involved: managers, developers and customers. This is achieved through consistent architectures, improved code quality of the output and a shorter time to market [69].

¹¹application logics are partly generated on the runtime

¹²most of the client side JavaScript code is generated from Java

4 Methods and materials

Recent RI Databank Service development plans described in chapter 2.1.1 result in significant challenges to the service's general architecture and components. Starting from 2018, the service will become a data provider and system integrator. For this stage real-life integrations with partners' CRIS systems were scheduled. Additionally, it was decided to take an attempt to replace Liferay framework with more lightweight alternatives.

The aim of the thesis is to address the challenges of the new service by developing a Proof of Concept (PoC) of the next major version of the application. The resulting prototype has to verify the compatibility of the selected tools and frameworks as well as to ensure service ability to process real CRIS API data. The two main objectives of the application are: translating incoming XML data into canonical format and ensuring the persistent storage of the result. The sample data for processing is prompted manually from one of the partner's CRIS (University of Helsinki). The project idea for the thesis originates from the author's experience working at CSC.

4.1 Methods

The research methodology of the thesis follows closely steps taken in the corresponding project. Thesis is a report of lessons learnt [70] from developing a prototype of RI Databank Service. The three main stages of the project are:

1. planning
2. implementation
3. results verification

During the first stage, both technical and non-technical requirements are gathered and analyzed. Based on the results, the functional component schema draft is developed. This subtask included arranging informal discussions with involved parties such as DB administrators along with colleagues responsible for the software architecture and de-

sign. Additionally, previous version of the service is comprehensively analyzed to avoid inheriting of any design faults and negative dependencies. On completion of the previous steps, a research of available technologies is performed to select the most suitable ones. The information is gathered from public sources such as frameworks' and tools' official documentation pages, source code and message boards.

Second stage, implementation, focuses on actual development of the prototype. The main goal of the stage is to ensure compatibility of the software components and to document possible drawbacks. As a part of the process, code generation functionality for several tools is trialed to fit the project's needs.

The last stage of the project is a verification of the results. This includes several types of testing, benchmarking the general performance indicators and asking for a formal approval of the superior colleagues.

The order of described phases does not strictly reflect the application development flow. This is caused by interdependencies of the stages. For example, during implementation phase, better alternatives might be found and used for the planned software components. Implementation and testing stages are overlapping, because unit testing is an inseparable part of the software development process.

4.2 Requirements and development environment

As a result of the planning phase, the following requirements were formulated, separated into functional and non-functional.

Functional requirements:

- input XML is processed into canonical format
- processed XML data objects are stored into a structured database
- stored data is exposed via REST API
- API is documented

Non-functional requirements:

- adaptability - ability to process various data in terms of the format and size
- flexibility - ability to adjust to internal data model modifications in the DB schema
- performance - data transformation average speed is at least 100 XML entities per second
- portability - application is platform-agnostic (unlike Liferay portlets)
- testability - service components are automatically testable

An additional constraint for the application development was usage of the Java technological stack, due to team members competences. Tools and development environment should be easy to set up, configure and maintain. Security of the application is excluded from the requirements as the goal of the project is to build a prototype.

The following tools, frameworks and technologies were used while developing the application:

- IntelliJ IDEA Ultimate IDE
- Java programming language version 8
- Spring Boot framework with DATA and REST extensions
- JavaEE JPA for Object-Relational Mapping (ORM)
- JAXB for XML processing support
- set of SpringFox libraries for Swagger API descriptions
- MapStruct entity mapping library
- JUnit unit testing framework
- Gradle application build tool
- XSLT for raw XML processing
- MySQL database for data persistent storage

To verify the conformity of the developed application with the requirements, several testing approaches were used. The testing was performed in a local development environment. Firstly, the stages of code generation were manually tested for the ability to compile and initiate application context (start Spring Boot). These trials were made using default application build tool Gradle. Service was started in IntelliJ IDEA development environment.

When developed component compiled without errors, its main logics were covered with unit tests. JUnit based tests included general functionality tests as well as tests measuring data processing performance. To assure that input data was successfully saved to persistent storage (database), manual testing with IDE built-in REST client was performed.

5 Solution

As a result of this thesis project, a prototype for the new version of RI Databank Service was created. The prototype is implemented as a Spring Boot¹³ application with MySQL database as a persistent data storage. Native Java Database Connectivity (JDBC) connector is used to transfer the data. Prototype expects a valid local or remote MySQL database installation with necessary read-write permissions. The application is build with the Gradle build tool, and unit tests are running during the build process. Although the test coverage tool (JaCoCo) is introduced, target coverage level is not defined.

Software is packaged as .JAR and requires Java Development Kit (JDK) version 8 or later to run in the hosting environment. Spring Boot has a build-in lightweight server functionality, as such, Web Service package does not require a separate Web Server for the deployment. The application is largely using Spring Boot default configurations. The source code is stored in CSC's internal git-based source code repository¹⁴.

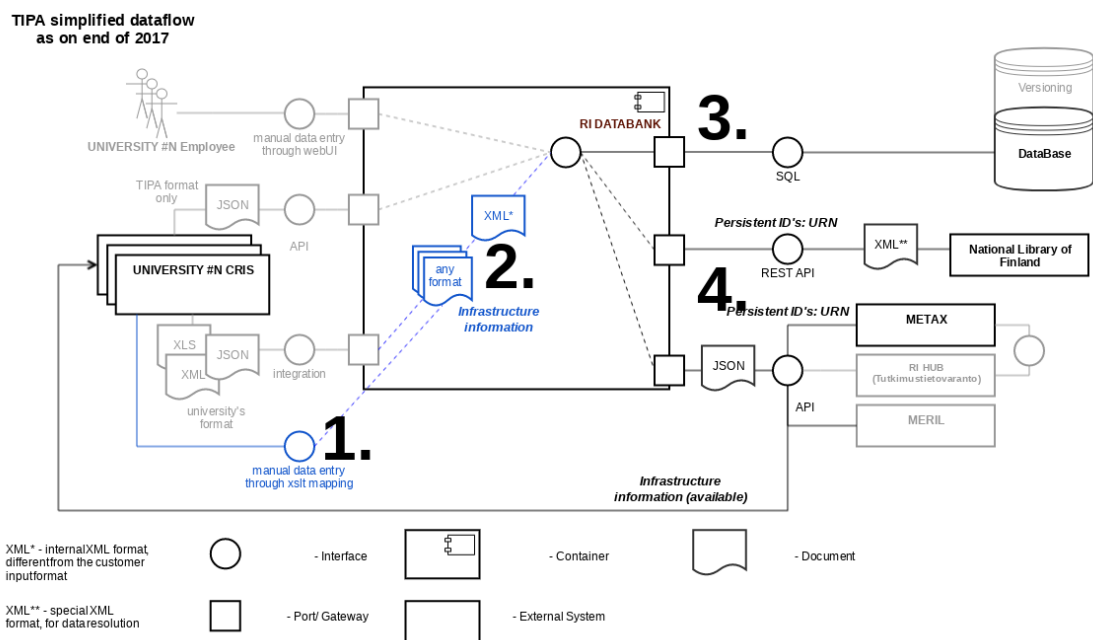


Figure 9: TIPA service data flow schema. Derived from CSC's service documentation (2017) [22]

¹³version 1.5.4.RELEASE

¹⁴available at <https://gitlab.csc.fi/ttv/tipa>

As displayed in figure 9, prototype data flow consists of the following stages:

1. manual query to partner's CRIS API; results of the query are fetched from the application source code bundle along with the specific call definition
2. input XML file is processed with the XSLT transformation to meet the requirements of the service canonical XML format
3. representation object of the canonical XML is stored to the database
4. entities stored in the database are retrieved via REST API

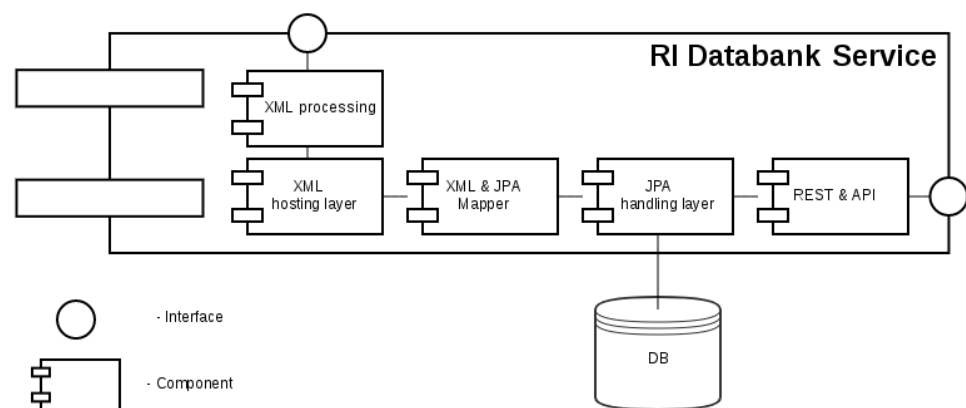


Figure 10: RI Databank Service component schema.

An overview of the architectural design of the prototype is presented in the figure 10. Application consists of the five conceptual modules and a database. XML processing module is responsible for the initial data transformation. It contains functionality for parsing the input XML data into its own canonical format and placing it into XML hosting layer classes. XML hosting layer is a part of the application data model and contains XML data representation classes. Mapper component is responsible for relation between XML hosting layer objects and JPA handling layer objects. It also provides declarations for data type conversions. JPA handling layer, as the name implies, is in charge of persisting and storing data into the database. JPA and XML handling layers form application data model layer. The last component is a REST API. This component utilizes the objects from JPA layer and exposes them as a Swagger-style described REST endpoints.

5.1 Data processing

XML data processing module combines both data handling key functionalities: data conversion from partner's XML into application internal format and following parsing of the output into Java code layer.

An important part of the data conversion functionality is implemented using XSLT stylesheet depicted in listing 1. It is responsible for matching the relevant fields of the input file and creation of the resulting canonical XML file. XSLT technology is suitable for handling input data transformation flexibly. Such processing can handle cases of lacking data, which are replaced with defaults, work with attributes and node values. Also, the technology requires very little extra configuration to operate.

```

1 <xsl:stylesheet version="3.0"
2     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3     xmlns:xalan="http://xml.apache.org/xalan">
4   <xsl:output method="xml" indent="yes" xalan:indent-amount="4"/>
5   <xsl:template match="/result">
6     <organizations>
7       <xsl:for-each select="externalOrganisation">
8         <organization>
9           <name>
10            <xsl:value-of select="name"/>
11          </name>
12          <customerId>
13            <xsl:value-of select="@uuid"/>
14          </customerId>
15          <country>
16            <xsl:choose>
17              <xsl:when test="address/country">
18                <xsl:value-of select='address/country' />
19              </xsl:when>
20              <xsl:otherwise>default</xsl:otherwise>
21            </xsl:choose>
22          </country>
23        </organization>
24      </xsl:for-each>
25    </organizations>
26  </xsl:template>
27 </xsl:stylesheet>

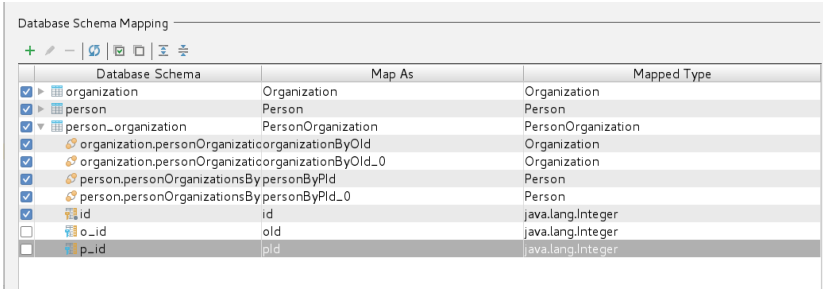
```

Listing 1: XSLT schema for Organization entity conversion [71]

Further XML internal format conversion into Java objects is realized in `Parser` class. Its methods are built to be generic¹⁵ and heavily utilize the features of Java XML processing APIs. Implemented methods use SAX parsing algorithm underneath. Alternatives of the overloaded methods¹⁶ for XML parsing `public E parseXml(File xml, ...)` allow to process data with or without validation against the target XSD schema. In order to be discoverable by Spring context, the described class is annotated as `@Service`.

5.2 Data models

The approaches and tools used for data model handling comply with the application flexibility requirement, defined in subsection 4.2. The main data model for the application is pushed to a database level making the SQL entity model a primary representation of the information structure. Other layers related to the model, JPA and XML representation layers, are derived from the application DB schema. This approach allows the application to retain flexibility for future data model changes.



Database Schema	Map As	Mapped Type
organization	Organization	Organization
person	Person	Person
person_organization	PersonOrganization	PersonOrganization
organization.personOrganizationByOld	Organization	Organization
organization.personOrganizationByOld_0	Organization	Organization
person.personOrganizationsByPersonByPld	Person	Person
person.personOrganizationsByPersonByPld_0	Person	Person
id	id	java.lang.Integer
o_id	old	java.lang.Integer
p_id	pld	java.lang.Integer

Figure 11: Configurations for generating JPA layer with IntelliJ IDEA. CSC (2018) [73]

The IntelliJ IDEA built-in features for code generation are widely used during the implementation of this module. The persistence support is handling the JPA layer generation and JAXB features create the XML hosting layer. To produce the results, the tools required minor to no adjustments to default configurations of code generation settings.

- 1 Caused by: org.hibernate.MappingException: Repeated column in mapping for entity: `fi.csc.tipa.model.jpa.PersonOrganization` column: `o_id` (should be mapped with `insert="false" update="false"`) ...

Listing 2: Error message caused by JPA layer generation misconfiguration

¹⁵a class or interface that is parameterized over object types

¹⁶a feature that allows a class to have more than one method having the same name, if their argument lists are different [72]

An example of the change to default configuration for the JPA layer establishment is illustrated by figure 11. As shown in the figure, it is required to disable the generation of foreign keys in the linking table representing many-to-many relation. Absence of this re-configuration causes service failure at the start-up stage with the error shown in listing 2.

```
1 @XmlElement
2 public class Organization { ... }
```

Listing 3: Annotation required for JAXB class recognition. CSC (2018) [74]

XML hosting layer was also generated by the IntelliJ IDEA, based on the classes resulting from the JPA layer creation phase. During this process, another significant issue was encountered. Resulting XML representation classes did not contain key annotation (`@XmlElement`) that helps JAXB with the class recognition. Annotation had to be manually added to the target object classes as shown on listing 3.

5.3 Mapping

To ensure smooth processing of data, JPA and XML Java objects had to be mapped to each other. This was completed using MapStruct mapping library. The library is abstracting away the verbose code creation to pre-compilation stage. As such, the mapping is relatively easy to implement. A `@Mapper` annotated interface was created for every pair of the entity classes with instantiation logics.

```
1 public interface DefaultTypeMapper {
2
3     fi.csc.tipa.model.jpa.Person personXmlToPersonJpa(Person personXml);
4
5     fi.csc.tipa.model.jpa.Organization
        organizationXmlToOrganizationJpa(Organization organizationXml);
6
7     fi.csc.tipa.model.jpa.PersonOrganization
        personOrganizationXmlToPersonOrganizationJpa(PersonOrganization
            personOrganizationXml);
8     ...
9 }
```

Listing 4: MapStruct JPA and XML entities mapping interface. CSC (2018) [75]

During the testing phase, it was noticed, that methods for type conversion have to co-locate inside the same interface, the `DefaultTypeMapper` interface was thus created.

Later it was extended by class-specific `@Mapper` annotated interfaces. This step allowed to keep the conceptually different code for JPA and XML entities separate and meet technical requirements of the library. Otherwise, the MapStruct did not require any extra configuration, was simple and fast to set up.

The benefit of using the described approach is a clear separation of concerns, where the actual code responsible for layers mapping is redone on every compilation and is therefore completely independent of the source and target classes' structures. This is only possible for cases, where the class fields are following similar naming patterns. Part of the code base for mapping the three entities to each other is shown in listing 4. The fields of classes are not present in the code being automatically matched by intelligent mapper. The only mapping-related limitation, faced in context of the work, is the incompatibility of the latest MapStruct library with the latest Spring Boot version¹⁷.

5.4 REST API

To generate a human-readable User Interface (UI), the prototype utilizes features of the SpringFox Swagger framework. Two interfaces, one for each exposed entity were created: `PersonRepository` and `OrganizationRepository`. Interfaces were configured by extending one of the JPA repositories¹⁸ and marking them with a common Spring `@RestResource` annotation. An `@EnableSwagger2` annotation had to be added to the Spring Boot application context class to enable the API description initialization. Additionally, a SpringFox configuration Bean was created and annotated as `@Bean`. As a result of these steps, the extendible skeleton for further API description was constructed. A set of HyperText Transfer Protocol (HTTP) access methods is generated for the empty interface by default. Swagger API description page contains UI block for trying out the methods. Part of UI with generated description and corresponding source code are presented in the figure 12.

The REST API implementation stage required a larger configuration effort. To properly

¹⁷MapStruct libraries: `org.mapstruct:mapstruct-processor:1.2.0.Final`, `org.mapstruct:mapstruct-jdk8:1.2.0.Final` and Spring Boot version `org.springframework.boot:spring-boot-starter-web:2.0.0.RELEASE` are incompatible as for March 2018

¹⁸`CrudRepository`, `PagingAndSortingRepository` Or `JpaRepository`

The image shows a side-by-side comparison of a Swagger API description and its source code. On the left, the Swagger UI displays two entities: 'Organization Entity' and 'Person Entity'. For 'Organization Entity', methods include GET /organizations (findAllOrganization), GET /organizations/{id} (findOneOrganization), DELETE /organizations/{id} (deleteOrganization), and GET /organizations/search/findByOrganizationId (find all Organizations that are associated with a given Person). For 'Person Entity', methods include GET /persons (findAllPerson), POST /persons (save a new Address), GET /persons/{id} (findOnePerson), PUT /persons/{id} (save a new Address), DELETE /persons/{id} (deletePerson), and PATCH /persons/{id} (save a new Address). On the right, the source code shows the implementation of these methods in Java, using annotations like @ApiOperation and @ApiResponse to describe the API endpoints.

Figure 12: Side-by-side presentation of the generated Swagger API description and originating source code

document Swagger API, every method should be extensively annotated¹⁹. Additional annotations are required in the model classes to prevent certain fields from disclosure (such as internal identifies). The Spring Boot REST component had to be prevented from exposing default "error" and "profile" controllers using regex²⁰. Also, an insufficient SpringFox Swagger framework documentation and the absence of relevant examples hindered the development process.

5.5 Testing

In order to verify the conformity of the application to the initial requirements, several types of tests were performed. The first group, manual compilation tests, originated from the nature of Java being a compiled language. After every code generation step or a significant configurational change, the application was rebuilt and started as a Spring Boot. This was done with IDE built-in tools and allowed to ensure that application is valid, at least from a formal configuration standpoint.

Next type of tests, unit tests, covered parsing, transformation and mapping functionalities. Two sources of data were used: dummy data, generated solely for the testing purposes,

¹⁹ @SuppressWarnings(UNCHECKED) @ApiOperation(FETCHES_ALL_THE_ORGANIZATIONS) simple method annotation example.

²⁰in theoretical computer science and formal language theory, a sequence of characters that define a search pattern [76]

and actual data from the partner's CRIS. The results of the unit tests have shown that selected tools and components(XSLT, XML MapStruct, JPA) integrate appropriately and are thus fit to purpose²¹.

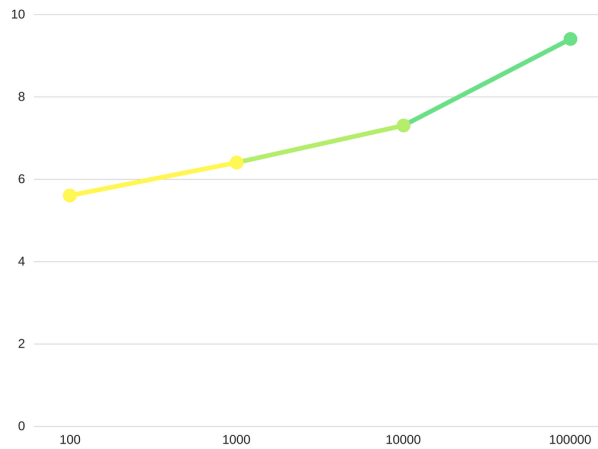


Figure 13: Results of real-life data XML processing

To ensure service accordance with quantitative requirements, XSLT processing performance tests were run as a part of the unit testing stage. The results of the performance tests are shown in figure 13. The graph uses a logarithmic scale, where horizontal axis displays the amount of entities per source document and the vertical axis - the natural logarithm of the mean time in milliseconds. The figure shows that significant amount of time is spent on establishing the `Transformer`²² object, while XML processing is performed fast, regardless of the size of the input file. Data conversion speed satisfies the requirements²³.

²¹ability of the commodity or service to do what it is designed for [77]

²²an object used to process XML from a variety of sources and write the transformation output to a variety of sinks [78]

²³non-functional requirement for data transformation average speed at least 100 XML entities per second is defined in section 4.2

6 Discussion

The result of the thesis project is a prototype of the next version of the RI Databank Service. In contrast with the previous version, the architecture of the current application is simpler, despite increase in the provided functionality. The prototype is able to handle various input data types and is thus better prepared for further integration iterations.

Among the conceptually suitable options for orchestrator based²⁴ integration architectures, Data Warehouse was chosen in favour of the Mediator model. Warehouse architecture provides opportunities to modify the data contents. Furthermore, it improves data quality by removing/aggregating data on the same entities and adding PIDs²⁵. Another reason for selecting the model is a common data storage. This is a technical requirement of the National Research Information Hub project²⁶: data should be kept in the same SQL storage in order to be queried directly from the DB by Extract, Transform, Load (ETL) Engine.

The decision to substitute Liferay container with the Spring Boot helped to reduce service architectural complexity and future maintenance burden. Another benefit of moving away from the Liferay framework is higher flexibility, obtained through easier customization of the components. For example, highly customizable SpringFox Swagger API description *versus* statically generated pages of the Liferay Service Builder API. Moreover, the decision significantly increased application portability across environments and platforms. This was achieved by decreasing technical requirements to the host system from the massive Liferay container to a lightweight and common Java Runtime Environment (JRE).

On the other hand, replacement of the powerful Service Builder framework of Liferay with several standalone components (JPA, JAXB, MapStruct) demanded integration efforts and might need deeper integration testing. Also, using multiple standalone components

²⁴orchestrator based models are more effective than non-orchestrated ones, as discussed in section 3.1

²⁵record linkage technique is a merging, that brings together information from two or more sources of data with the object of consolidating facts concerning an individual or an event that are not available in any separate record [79]

²⁶an umbrella project for RI Databank Service

may cause problems with potential versions incompatibility. On the smaller scale, this issue was already encountered at the implementation stage, where the latest Spring Boot version could not be used, because of the missing MapStruct mapping library support. However, upgrading Service Builder framework is inseparable from upgrading the Liferay's version, which may result in more complications and higher testing efforts²⁷.

The benefits of the new service architecture would be hardly achievable without an intense use of the code generation tools. JAXB and JPA allowed a generation of clean, separable and extensible code layers. Code generation time is negligible compared to the effort spent on the tools' configurations. Further code regeneration for extended modelling and developing production-grade application should not introduce any new obstacles, but would still require certain amount of manual calibration. In general, utilization of code generation tools has accelerated development of the application, as confirmed in section 3.4. Nonetheless, generation of complex, multiple-level-nested entities, with JPA and JAXB was not tried in scope of this work. As follows, further prototyping with such entities is required along with a research on the related XSLT performance.

The XSLT technology proved to be simple and effective in use. Being an older technology, it had most of the relevant use cases covered and produced no bugs during the development and testing stages. Unfortunately, direct reuse of the XSLT transformation created for the prototyping purposes is not possible for further application development. A new schema must be created for every new partner's data source. As such, only the experience is transferable. Even so, the XSLT usage shifts the data mapping and conversion outside Java code layers and simplifies thus the architecture in accordance with the separation of concerns principle.

Modern IDEs provide extensive support for code generation tools and frameworks. However, exclusive reliance on the IntelliJ IDEA Ultimate might impose certain limitations on the process. Limitations include a need for a paid license, as open-source version does not support most of the used tools, and a natural vendor lock-in. Moreover, active usage of the IDE UI and build-in tools (Gradle, DB management tool) during the development, may lead to complications in the process automation, for example Continuous Integration and Continuous Deployment (CI/CD) establishing.

²⁷ all the other AVAA applications would also require additional testing, because of using the same Liferay container, as described in section 2.1.2

Considering design and implementation options for mapping component, MapStruct library was chosen. Being a pre-compilation type tool, it should theoretically provide a fast entity mapping²⁸. This helped to preserve the processing speed of the layer at the acceptable level.

The resulting prototype has a rich REST API. It can act as a base for establishing UI layer with any relevant front-end technology. This allows higher flexibility in the service ecosystem so that the RI Databank Service and Research Information Hub could use it as a base for UI layer creation. Additionally, the SpringFox's generated API description Web pages could be used to interactively prompt the data via UI on their own. However, due to a relatively tight coupling of the SpringFox library with the REST repositories layer, API can not be externalized to a separate application, which could increase modularity.

Another important observation concerns the input data processing speed, which was set as a non-functional requirement. Prototyping phase has shown, that there is no demand for a faster than XML processed formats (like JSON) because of the satisfactory results of performance testing. Nevertheless, the scope of the prototype did not include processing of JSON as a source format. Additional tests should be performed in order to validate format processing support with the XSLT version 3.0.

The developed version of the service could have several improvements. For example, the application could fetch the API descriptions from an external source. The source could be any collaborative tool²⁹, wherein the data is collected by a plugin. This approach would allow non-technical team members (analysts, product owners) to edit the descriptions externally. Separating API descriptions from the source code would remove description maintenance burden from the developers. Also, to ease the regeneration process, a script to add missing annotations to certain generated classes could be created. Moreover, the current version of the prototype could clearly have higher test coverage for data conversion and mapping components' methods.

Next planned step of the project is extending the prototype's skeleton to accommodate the variance of the real data sets from partners' CRISs. Additionally, more tests must be created to cover mapping and parsing functionalities. One more important step to-

²⁸for comparison of mapping approaches see chapter 3.3

²⁹for example Google Spreadsheet

wards the production-grade application would be configuring CI/CD pipelines for testing with more data (including more complex entities) and load. To conceptually finalize the project, a mechanism for deduplication³⁰ of data received during integrations, should be developed. These steps will increase operational stability of the service and offer the end users added value by providing a homogeneous view on Finnish RIs.

The implemented application is strictly following the functional requirements defined earlier in section 4.2. The prototype is satisfying most of the non-functional requirements as well. The only requirement that is met partially is adaptability, which refers to the application's ability to adjust to various input data formats. The prototype's design assumes this feature, however, tests were performed for single format only (XML). The prototype proves feasibility of the solution, it shows that chosen tools and technologies allow to meet the project goals.

The result of the thesis is based on the real job task and provides therefore clear interest for the employer. Developed PoC will shape the base for implementing the key software component for the RI Databank Service project.

Due to a small team size, I acted both as a developer and the Scrum Master³¹ for the project. This resulted in a significant development of soft and hard skills. At times, stakeholders required assistance and guidance in shaping the project requirements. Helping them in this process advanced the understanding of the service business domain and its values for the customers. Additionally, documenting and managing the requirements improved communication and technical writing skills. Not to mention, working on the prototype had a positive impact on the team-work skills. Moreover, I have learnt better integration techniques using XSLT and JAXB. In addition, I got acquainted with sophisticated code generation tools, identified nested advantages and disadvantages. Implementing the JPA layer polished my existing programming skills. Furthermore, the knowledge of the used tools' (IDE, build tool) functionalities has increased.

³⁰ other names for deduplication are "record linkage" and "data linkage"

³¹ Scrum Master is a person who, among other responsibilities, helps to improve interactions to maximize the value created by the Scrum Team [80]

7 Conclusion

Due to changes in its development context, the RI Databank Service application faced new functional requirements. As a response to this, a prototype for a new version of the service had to be developed. The prototype implementation was completed as a work task at CSC. The most important requirements for the application were the ability to integrate with partner systems and preserve the overall architectural flexibility. The prototype has benefitted from the application's previous version as well as the best practices of the software design. The created software actively utilized the concept of the code generation which in turn accelerated development process.

During the implementation phase an overall approach viability analysis was completed. In addition, the technological compatibility of the software components' verification was performed. This was achieved by comprehensive manual, unit and performance testing. One limitation of the project is an absence of additional testing in production-like environments. This includes tests of processing the full variety of actual data, load testing and configuring CI/CD pipelines.

The results show that the developed application is able to integrate with common CRIS data format and capable of adjusting to the dynamic data model. Performance testing stage has confirmed that data transformation speed satisfies the minimal speed requirements. In contrast with the previous version, the current application architecture is simpler, despite the overall increase in its functionality. Built prototype will be used as a base for further RI Databank Service project development at CSC.

Based on the results of this thesis, further study of the code generation tools compatibility for integration purposes might be performed. It would be also interesting to investigate the options and potential approaches for a record deduplication problem. The research should aim at scenarios, where identifiers of the data entities, originating from various sources are missing or non-unique.

Bibliography

- 1 Oracle. Packaging Programs in JAR Files; 2018. Available from: <https://docs.oracle.com/javase/tutorial/deployment/jar/index.html> [cited 2018-04-30].
- 2 Pivotal. Understanding WAR; 2018. Available from: <https://spring.io/understanding/WAR> [cited 2018-04-30].
- 3 Oracle. Java™ Programming Language; 2018. Available from: <https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html> [cited 2018-04-30].
- 4 Agafonkin V. Leaflet - a JavaScript library for interactive maps; 2017. Available from: <https://leafletjs.com/> [cited 2018-04-30].
- 5 Techopedia. What is Linked Data? - Definition from Techopedia; 2018. Available from: <https://www.techopedia.com/definition/5179/linked-data> [cited 2018-04-30].
- 6 Google. Portlet Definition; 2018. Available from: <https://www.google.fi/search?q=portlet&oq=portlet&aqs=chrome..69i57j69i65j0j69i60l2j0.1059j1j4&sourceid=chrome&ie=UTF-8> [cited 2018-04-30].
- 7 Liferay. What is Service Builder? - Liferay Developers Network; 2018. Available from: https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/what-is-service-builder [cited 2018-04-30].
- 8 Pivotal. Introduction to Spring Framework ; 2018. Available from: <https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html> [cited 2018-04-30].
- 9 Pivotal. Spring Boot ; 2018. Available from: <https://projects.spring.io/spring-boot/> [cited 2018-04-30].
- 10 Techopedia. What is a Web Service? - Definition from Techopedia; 2018. Available from: <https://www.techopedia.com/definition/25301/web-service> [cited 2018-04-30].
- 11 CSC. About CSC; 2016. Available from: <https://www.csc.fi/csc> [cited 2018-03-29].
- 12 Open Science and Research Initiative. Research Infrastructure Databank Service Report 2016; 2016. Available from: <https://openscience.fi/documents/10864/62537/20160823+Open+science+services-TIPA.pptx>.
- 13 Research Infrastructure Database Service. Research Infrastructures / About; 2017. Available from: <http://infras.openscience.fi/#!about> [cited 2017-12-25].

- 14 Open Science and Research Initiative. Data bank on RI; 2017. Available from: <https://openscience.fi/data-bank-on-ri> [cited 2017-12-25].
- 15 CSC. AVAA - About AVAA; 2017. Available from: <https://avaa.tdata.fi/web/avaa/tietoa-palvelusta> [cited 2018-03-16].
- 16 Academy of Finland, Finnish Research Infrastructures, Ministry of Education and Culture. Finland's strategy and roadmap for Research Infrastructures 2014–2020. 2014; Available from: http://www.aka.fi/globalassets/awanhat/documents/firi/tutkimusinfrastruktuurien_strategia_ja_tiekartta_2014_en.pdf.
- 17 Open Science and Research Initiative. Research Infrastructure Databank Service - Persistent Identifiers; 2017. Available from: <https://openscience.fi/ri-pid> [cited 2017-12-25].
- 18 CSC. Research Information Hub - Eduuni-wiki; 2018. Available from: <https://wiki.eduuni.fi/display/CSCTTV/In+English> [cited 2018-03-07].
- 19 Research Information Hub. Research Information Hub – Towards the Finnish Research Information Hub 2020-; 2018. Available from: <https://research.fi/> [cited 2018-03-07].
- 20 Oracle. Java Rich Internet Application Patterns; 2018. Available from: <https://docs.oracle.com/javase/7/docs/technotes/guides/jweb/index.html> [cited 2018-03-16].
- 21 CSC. TIPA Project older version source code; 2017. Available from: <https://github.com/avaa-csc/avaa-tipa>.
- 22 CSC. TIPA schemes - Eduuni Wiki; 2017. Available from: <https://wiki.eduuni.fi/display/att/TIPA+schemes>.
- 23 AVAA. JSONWS API; 2017. Available from: <https://avaa.tdata.fi/api/jsonws?contextPath=/tupa-portlet&signature=%2Ftupa-portlet%2FHelper%2Fget-capabilities-0> [cited 2017-12-25].
- 24 AVAA. REST API. 2017; Available from: <https://avaa.tdata.fi/tupa-portlet/api/infraservice/infra-URN-URL-identifier-location.xml>.
- 25 CSC. TIPA IOW data model; 2017. Available from: <http://iow.csc.fi/model/tipa/> [cited 2017-12-25].
- 26 Kainulainen A. Suomen tutkimusinfrastruktuurit - palvelun tiedonsiirron pilotointi; 2017.
- 27 Elsevier. Pure Hosted Edition; 2014. Available from: https://www.elsevier.com/___data/assets/pdf_file/0004/53437/pure-hosted-edition-brochure-version2.01August2014.pdf.
- 28 Elsevier. Pure | Helps Research Managers at your Institution; 2018. Available from: <https://www.elsevier.com/solutions/pure> [cited 2018-01-19].
- 29 Elsevier. Elsevier Pure Brochure; 2016.

- 30 Elsevier. Features - Pure | Elsevier; 2018. Available from: <https://www.elsevier.com/solutions/pure/features> [cited 2018-01-19].
- 31 University of Helsinki PURE System. PURE API longer descriptions; 2018. Available from: https://tuhat.helsinki.fi/ws/api/510/api-docs/documentation/Content/Topics/Web_Services_Intro.htm [cited 2018-01-19].
- 32 IBM. Data Integration; 2018. Available from: <https://www.ibm.com/analytics/data-integration> [cited 2018-03-13].
- 33 Techopedia. What is Data Integration? - Definition from Techopedia; 2018. Available from: <https://www.techopedia.com/definition/28290/data-integration>.
- 34 Gartner. Data Integration - Gartner IT Glossary; 2018. Available from: <https://www.gartner.com/it-glossary/data-integration-tools> [cited 2018-01-18].
- 35 Ullman J. Information Integration Mediators Warehousing Answering Queries Using Views; 2007. Available from: <http://infolab.stanford.edu/~ullman/fcdb/aut07/slides/integration.pdf>.
- 36 IPUMS. IPUMS; 2017. Available from: <https://www.ipums.org/whatIsIPUMS.shtml> [cited 2018-01-18].
- 37 Gagnon M. Ontology-Based Integration of Data Sources. 2007; Available from: http://fusion.isif.org/proceedings/fusion07CD/Fusion07/pdfs/Fusion2007_1318.pdf?
- 38 Bell R. A beginner's guide to Big O notation ; 2018. Available from: <https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/> [cited 2018-04-30].
- 39 Ferreira DR. Enterprise systems integration: A process-oriented approach. Springer-Verlag; 2013. Available from: <https://link-springer-com.libproxy.aalto.fi/content/pdf/10.1007%2F978-3-642-40796-3.pdf>.
- 40 Fong JSP. Information Systems Reengineering, Integration and Normalization. Springer Publishing; 2015. Available from: <https://link-springer-com.libproxy.aalto.fi/content/pdf/10.1007%2F978-3-319-12295-3.pdf><http://link.springer.com/10.1007/978-3-319-12295-3>.
- 41 Google. Semantic Web - Google Search; 2018. Available from: https://www.google.fi/search?ei=QijXWuboO8W3swH9zY2wBA&q=semantic+web&oq=semantic+web&gs_l=psy-ab.3..35i39k1j0i20i263k1j0l8.2175.2175.0.2588.1.1.0.0.0.70.70.1.1.0....0...1c.1.64.psy-ab..0.1.70....0.ljiPURlwONg [cited 2018-03-15].
- 42 W3C. XML Introduction; 2018. Available from: https://www.w3schools.com/xml/xml_what.asp [cited 2018-01-21].
- 43 W3C. Extensible Markup Language (XML); 2015. Available from: <https://www.w3.org/XML/> [cited 2018-01-21].

- 44 Doan A, Halevy A, Ives Z. Principles of Data Integration - 15. Morgan Kaufmann; 2012. Available from: <http://www.sciencedirect.com/science/article/pii/B9780124160446000156>.
- 45 Tutorials Point. XML Overview (benefits); 2018. Available from: https://www.tutorialspoint.com/xml/xml_overview.htm [cited 2018-01-21].
- 46 Oracle. Why StAX? (The Java™ Tutorials; Java API for XML Processing (JAXP); Streaming API for XML); 2017. Available from: <https://docs.oracle.com/javase/tutorial/jaxp/stax/why.html> [cited 2018-01-21].
- 47 Nurseitov N, Paulson M, Reynolds R, Izurieta C. Comparison of JSON and XML Data Interchange Formats: A Case Study. Scenario. 2009;59715:1–3. Available from: <http://www.cs.montana.edu/izurieta/pubs/caine2009.pdf>.
- 48 Haq ZU, Khan GF, Hussain T. A Comprehensive analysis of XML and JSON web technologies. New Developments in Circuits, Systems, Signal Processing, Communications and Computers. 2014;p. 102–109. Available from: <http://www.inase.org/library/2015/vienna/bypaper/CSSCC/CSSCC-14.pdf>.
- 49 Oracle. Java Architecture for XML Binding (JAXB); 2003. Available from: <http://www.oracle.com/technetwork/articles/javase/index-140168.html> [cited 2018-02-23].
- 50 Webopedia. What is data marshalling? Webopedia Definition; 2018. Available from: https://www.webopedia.com/TERM/D/data_marshallng.html.
- 51 Fialli J. The Java™ Architecture for XML Binding (JAXB) Java™ Architecture for XML Binding (JAXB) Specification. 2003;Available from: http://download.oracle.com/otn-pub/jcp/7196-jaxb-1.0-fr-spec-oth-JSpec/jaxb-1_0-fr-spec.pdf.
- 52 W3C. XSL Introduction; 2018. Available from: https://www.w3schools.com/xml/xsl_intro.asp [cited 2018-02-23].
- 53 W3C. Transformation (XSLT) - W3C; 2015. Available from: <https://www.w3.org/standards/xml/transformation> [cited 2018-02-23].
- 54 W3C. XSL Transformations (XSLT) Version 3.0; 2017. Available from: <https://www.w3.org/TR/2017/REC-xslt-30-20170608/> [cited 2018-01-21].
- 55 Tidwell D. XSLT. O'Reilly Media; 2008. Available from: https://books.google.fi/books?hl=en&lr=&id=VZaiiPkH94sC&oi=fnd&pg=PR7&dq=xslt&ots=03HrwiNJUu&sig=5oOvQoQJ858O9dk94H2C4i8WwJE&redir_esc=y#v=onepage&q=xslt&f=false.
- 56 Google. Mapping Definition - Google Search; 2018. Available from: https://www.google.fi/search?ei=CVGqhttps://www.google.fi/search?ei=CVGqWqLwFYOA6QTSzYvgBg&q=mapping+definition&oq=mapping+definition&gs_l=psy-ab.3..0i71k1l8.0.0.0.71382.0.0.0.0.0.0.0.0....0...1c..64.psy-ab..0.0.0....0.Lxt_7Ztxz4Q [cited 2018-03-15].
- 57 Techopedia. What is Data Quality? - Definition from Techopedia; 2018. Available from: <https://www.techopedia.com/definition/6750/data-mapping>.

- 58 Oracle. Trail: The Reflection API; 2015. Available from: <http://docs.oracle.com/javase/tutorial/reflect/index.html> [cited 2018-03-15].
- 59 Oracle. Using Java Reflection; 1998. Available from: <http://www.oracle.com/technetwork/articles/java/javareflection-1536171.html>.
- 60 Hombergs T. Robust Java Object Mapping With Minimal Testing Overhead Using reMap - Reflecting; 2017. Available from: <https://reflecting.io/autotmatic-refactoring-safe-java-mapping/> [cited 2018-03-15].
- 61 MapStruct. MapStruct – Java bean mappings, the easy way!; 2017. Available from: <http://mapstruct.org/> [cited 2018-03-15].
- 62 Wyszomierski M. Mapping Dozer vs MapStruct; 2016. Available from: <http://mariusz.wyszomierski.pl/en/mapping-dozer-vs-mapstruct/> [cited 2018-03-15].
- 63 Techopedia. What is Code Generation? - Definition from Techopedia; 2018. Available from: <https://www.techopedia.com/definition/6531/code-generation>.
- 64 Fowler M. Code generation for dummies. Methods & Tools. 2009;p. 65–89. Available from: <http://www.methodsandtools.com/archive/archive.php?id=86>.
- 65 Liferay. What is Service Builder? - Liferay 6.2 - Liferay Developer Network; 2017. Available from: https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/what-is-service-builder [cited 2018-03-07].
- 66 Oracle. Java Persistence API; 2018. Available from: <http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html> [cited 2018-03-15].
- 67 Oracle. Java Persistence/What is JPA?; 2017. Available from: https://en.wikibooks.org/wiki/Java_Persistence/What_is_JPA%3F [cited 2018-03-15].
- 68 Williams NS. Professional Java® for Web Applications. John Wiley & Sons; 2014.
- 69 Herrington J. Code Generation in Action. Manning; 2003. Available from: <https://www.manning.com/books/code-generation-in-action>.
- 70 Bjørnson FO, Dingsøyr T. Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. Information and Software Technology. 2008;50(11):1055–1068. Available from: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.502.5306&rep=rep1&type=pdf>.
- 71 CSC. TIPA XSLT Stylesheet source code; 2018. Available from: <https://gitlab.csc.fi/ttv/tipa/blob/poc/src/test/resources/HelsinkiUniversity/initial/ext-orgS.xslt> [cited 2018-03-21].

- 72 BeginnersBook. Method Overloading in Java with examples; 2018. Available from: <https://beginnersbook.com/2013/05/method-overloading/> [cited 2018-04-30].
- 73 CSC. TIPA Project new version source code; 2018. Available from: <https://gitlab.csc.fi/ttv/tipa> [cited 2018-03-21].
- 74 CSC. TIPA Organization Class source code; 2018. Available from: <https://gitlab.csc.fi/ttv/tipa/blob/poc/src/main/java/fi/csc/tipa/model/xml/Organization.java#L43-44> [cited 2018-03-21].
- 75 CSC. TIPA DefaultMapper Interface source code; 2018. Available from: <https://gitlab.csc.fi/ttv/tipa/blob/poc/src/main/java/fi/csc/tipa/service/mappers/DefaultTypeMapper.java> [cited 2018-03-21].
- 76 Google. Regular Expression Definition; 2018. Available from: https://www.google.fi/search?ei=R9juWpDYOYOUsgG85a_ADQ&q=regular+expression+definition&oq=regular+expression+definition&gs_l=psy-ab.3..0i7i30k1l10.4920.8459.0.8752.17.17.0.0.0.124.1047.14j1.15.0....0...1c.1.64.psy-ab..3.14.990....0.att68VmBit4 [cited 2018-04-30].
- 77 WikiBooks. ITIL v3 (Information Technology Infrastructure Library)_Service Operation - Wikibooks, open books for an open world; 2017. Available from: [https://en.wikibooks.org/wiki/ITIL_v3_\(Information_Technology_Infrastructure_Library\)/Introduction](https://en.wikibooks.org/wiki/ITIL_v3_(Information_Technology_Infrastructure_Library)/Introduction) [cited 2018-03-11].
- 78 Oracle. Transformer (Java Platform SE 8); 2014. Available from: <https://docs.oracle.com/javase/8/docs/api/javax/xml/transform/Transformer.html> [cited 2018-03-24].
- 79 OECD. OECD Glossary of Statistical Terms; 2006. Available from: <http://stats.oecd.org/glossary/detail.asp?ID=3103> [cited 2018-03-25].
- 80 Scrum Alliance. The Scrum Guide - Scrum Alliance; 2014. Available from: <https://www.scrumalliance.org/learn-about-scrum/the-scrum-guide> [cited 2018-03-28].