Deliverables No. 3.5

Final System Architecture

Grant Agreement No.:	270089
Deliverable No.:	D3.5
Deliverable Name:	Final System Architecture
Contractual Submission Date:	31/01/2015
Actual Submission Date:	31/01/2015

Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

This project has received funding from the European Union's Seventh Programme for research, technological development and demonstration under grant agreement No 270089.

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	p-medicine
Project Full Name:	From data sharing and integration via VPH models to personalized medicine
Deliverable No.:	D3.5
Document name:	Final System Architecture
Nature (R, P, D, O)[1]	R
Dissemination Level (PU, PP, RE, CO)[2]	RE
Version:	1.0
Actual Submission Date:	31/01/2015
Editor: Institution: E-Mail:	Stelios Sfakianakis FORTH-ICS ssfak@ics.forth.gr

ABSTRACT:

This document presents the final design for the p-medicine architecture. We describe the generic context of personalized medicine and the specific application domains of the p-medicine platform. The different architectural elements are then classified into these application domains and their interfaces and interactions are fully described based on the input of the other work packages.

KEYWORD LIST: architecture, domain driven design, programmatic interfaces, REST

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 270089.

[1] **R**=Report, **P**=Prototype, **D**=Demonstrator, **O**=Other

[2] **PU**=Public, **PP**=Restricted to other programme participants (including the Commission Services), **RE**=Restricted to a group specified by the consortium (including the Commission Services), **CO**=Confidential, only for members of the consortium (including the Commission Services)

MODIFICATION CONTROL			
Version	Date	Status	Author
0.1	13/03/2014	Draft	Stelios Sfakianakis, FORTH
0.2	22/04/2014	Draft	Stelios Sfakianakis, FORTH
0.4	23/04/2014	Draft	Stelios Sfakianakis, FORTH
0.5	30/07/2014	Draft	Stelios Sfakianakis, FORTH
0.6	25/01/2015	Pre-final	Stelios Sfakianakis, FORTH
1.0	31/01/2015	Final	Stelios Sfakianakis, FORTH

List of contributors

- Stelios Sfakianakis, FORTH
- Alberto Anguita, UPM
- Elias Neri, Custodix
- Jordi Mares, UCL
- Juliusz Pucacki, PSNC
- Gabriele Weiler, FhG-IBMT
- George Zacharioudakis, FORTH
- Manolis Tsiknakis, FORTH
- Aisan Maghsoodi, Philips
- Xaris Kondylakis, FORTH

Contents

1 EXECUTIVE SUMMARY 6
2 INTRODUCTION 7
2.1 CHANGES SINCE DELIVERABLE 3.2 7
2.2 THE P-MEDICINE DOMAIN 7
2.3 STAKEHOLDERS AND CONCERNS 9
2.4 A LAYERED VIEW OF THE FINAL ARCHITECTURE 10
3 THE STRUCTURE OF THE P-MEDICINE SYSTEM 13
3.1 AUTHENTICATION AND AUTHORIZATION 14
3.2 DATA MANAGEMENT 16
3.2.1 Initial Upload and the Pseudonymization framework 16
3.2.2 Ontology Annotation and Translation 17
3.2.3 Data Warehouse 18
3.3 COMPUTATIONAL CANCER MODELLING 20
3.4 PATIENT EMPOWERMENT 21
3.4.1 The ALGA-C questionnaire 22
3.4.2 Personal Health Record 22
3.4.3 Patient Empowerment support services: Drug-Drug interactions 23
3.4.4 Consent Services 23
3.5 CLINICAL DECISION SUPPORT 25
3.5.1 The p-medicine workbench 26
4 TECHNICAL DESCRIPTION OF THE COMPONENTS 29
4.1 DATA WAREHOUSE 29
4.2 TTP SERVICES 35
4.3 ONTOLOGY ANNOTATOR 36
4.4 DATA TRANSLATOR 37
4.5 ALGA-C PROFILING SERVICE 40
4.6 DRUG – DRUG INTERACTION SERVICE 41
4.7 “DONOR’S TOOL” CONSENT ACCESS SERVICE 43
4.8 WORKBENCH 45
5 NON FUNCTIONAL REQUIREMENTS 51
6 DEPLOYMENT 52
7 ARCHITECTURE COMPLIANCE 55
8 CONCLUSIONS 58
9 REFERENCES 59
APPENDIX A – FORMAL SCALABILITY ANALYSIS IN THE P-MEDICINE PLATFORM 60
APPENDIX B - ABBREVIATIONS AND ACRONYMS 81

Figures

Figure 1 The main "context" or application areas of the personalized medicine 8
Figure 2 The p-medicine stakeholders layered 9
Figure 3 A layered view of the p-medicine architecture...10
Figure 4 The p-medicine architecture divided in three operational contexts..........................11
Figure 5 The colour coding introduced by Coad and used throughout this document 13
Figure 6 A domain centred view of the architecture 14
Figure 7 The XACML based Attribute Based Access Control model....................................15
Figure 8 The users management domain model..15
Figure 9 The anonymization and ontology annotation steps in the data upload process 16
Figure 10 The ontology annotation and translation processes that take place in the p-medicine zone 17
Figure 11 The "domain" of the Data Warehouse..18
Figure 12 The abstract interface of the Data Warehouse...20
Figure 13 The computational modelling domain 21
Figure 14 The Simulation Service abstract interface..21
Figure 15 The ALGA-C Questionnaire model 22
Figure 16 The ALGA-C Profiling abstract interface 22
Figure 17 The drugs and their interactions 23
Figure 18 The abstract interface of the Drug Interaction service 23
Figure 19 The main entities in the e-Consent context 24
Figure 20 Overview of the interactions and the components in the e-Consent services........24
Figure 21 The consent documents access interface..25
Figure 22 The CDS Framework 's architecture 26
Figure 23 The architecture of the p-medicine workbench 27
Figure 24 The domain model of the p-medicine workbench..27
Figure 25 The interface of the workbench..28
Figure 26 The deployment for the prototype system..52
Figure 27 The levels of the architecture conformance according to TOGAF 55

1 Executive Summary

The p-medicine project aims to design and build a platform for personalized medicine so its grand vision is to revolutionize today's medical practice to become more proactive and better targeting the right patient the right time. From a technical point of view the p-medicine platform is a system consisting of various collaborating subsystems that support different aspects of the personalized medicine vision and therefore it presents a certain system architecture. Generally speaking the architecture of a system is about its structure and the vision for the objectives and the goal that the final product will serve, and therefore it's always present even if not documented at all.

This document intends to provide such documentation and to present the final crystallization of the p-medicine system architecture. Emphasis is given on the application areas and the context specific scenarios that present a showcase for the personalized medicine. At the same time we have strived to provide the description of the underlying components, interactions, and technical requirements in a concise and terse way and provide references to the work done in other work packages and reported in their deliverables.

2 Introduction

2.1 Changes since Deliverable 3.2

In the Deliverable 3.2 “Initial System Architecture” we have followed a principled design based on the ISO/IEC 42010:2007 standard and the use of the Rozanski and Woods set of viewpoints [1]. Since then we have made the following simplifications and modifications:

- The functional and information views have been merged. The functional view documents the system’s functional elements, their responsibilities, interfaces, and primary interactions while the information view “architecture stores, manipulates, manages, and distributes information”. It then became apparent that one of the most important functionality of the system is the management of information and it’s so much pervasive into the rest of the functional characteristics of the system that presenting it as a different view does no justice to it. So now the conceptual methodology for describing the architecture is based on the identification of its core “domain models” that encapsulate the data and the data-related processing tasks, effectively making the data and the information extracted from it the basic driving force for the definition of the architecture. The relevant paragraph is “The p-medicine domain” starting on page 7.
- The concurrency view which describes the concurrency structure of the system and maps functional elements to concurrency units (e.g. threads) was considered to be too low level for the design of a large, distributed system as the one introduced by the p-medicine project. History in distributed systems has shown that the use of low level and fine –grained constructs such as locking mechanisms (semaphores, monitors, etc.), shared state, and transactions are not scalable and certainly a faulty basis for the design of “correct”, heterogeneous, network accessible and operated systems[2]. Instead, we emphasize the use of coarse-grained message exchange patterns across the boundaries of “services” as described in Section 3 starting on page 13 of this document.
- The deployment view (chapter 6) has been extended to include information about the hardware requirements of the p-medicine software components based on the extensive stress testing and benchmarking tasks that the technical partners performed.
- A new chapter 7 provides details about the compliance of third party or newly developed tools with the p-medicine architecture and the integration means that bring the whole platform together.

In the following sections we try to provide a clear description of the envisaged architecture. We start by describing the *domain* of the p-medicine as an exemplar system (or better, group of systems) for personalized medicine.

2.2 The p-medicine domain

All good design starts with a “divide-and-conquer” (partitioning) method following by the abstraction of the underlying entities into more general and less detailed ones, which allows us to manage the complexity of the problem at hand. The need for this design strategy was recognized early on, based on the user needs and scenarios (**Deliverable 2.2**) In this line of thought, we can identify the following generic areas or “functional profiles” where the p-medicine platform aims at building solutions towards the goal of personalized medicine:

- *Patient Empowerment*, where the patient actively participates and interacts with the system in order to become aware of new possibilities for improving his health or helping the active research, like searching for clinical trials to enrol in.

- *Knowledge discovery, exploratory, and predictive analysis and data mining*. This incorporates scenarios like new biomarkers discovery and experimentation in order to produce new knowledge that of course needs to be subsequently validated.
- *Data management*, which deals with the handling of data from their initial import to all the stages of their "life cycle" by maintaining linkage and provenance. This area also incorporates semantic harmonization tools, which are responsible for semantic annotation, translation, ontology maintenance, etc.
- *Computational Cancer Modelling*, where tools and components supporting the modeling and simulation of tumor growth and response to drugs and other therapy plans are located.
- *Clinical Decision Support*, which provide software tools and systems that support physicians in decision making in their daily care of patients.

These application areas constitute "islands of functionality" for the personalized medicine vision of the project and are depicted graphically in the following figure (Figure 1). At the same time we are trying to implement them as "*bounded contexts*", a term borrowed from the seminal work of Eric Evans (and others) as described in his "Domain Driven Design" book [3]. The Domain Driven Design (DDD) methodology focus on the modelling of the core domain of an application or system using the language agreed with the domain experts. We believe this is a useful methodology to follow because the p-medicine platform aims to support a broad range of applications for personalized medicine and the complexity of such diverse domain cannot be easily addressed by a single viewpoint.

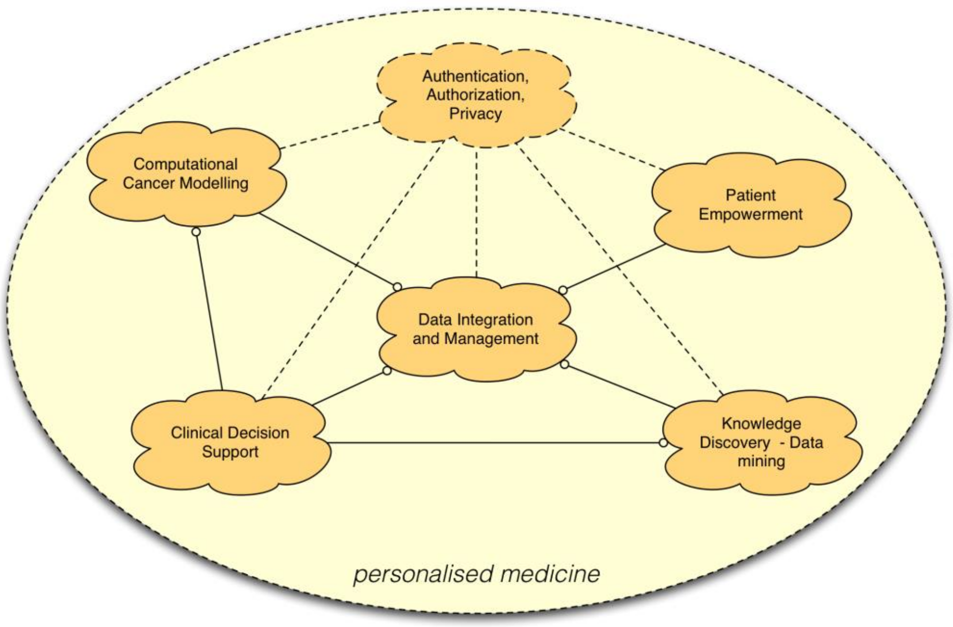

Authentication, Authorization, Privacy
Computational Cancer Modelling
Patient Empowerment
Data Integration and Management
Clinical Decision Support
Knowledge Discovery - Data mining
personalised medicine

Figure 1 The main "context" or application areas of the personalized medicine

In the figure above we show the main contexts (or sub-domains) of personalized medicine and their possible interactions. Central to this view is the Data Management domain that is used by all the other contexts. Other important interconnections are the use of Knowledge Discovery -Data Mining and the Computational Cancer Modelling domains by the Clinical Decision Support domain, underlining the requirement that the research outcomes in prediction, simulation and visualization can be used as support means for better diagnosis or individualization therapy in the clinical domain. Supportive for all the other contexts is the Security related subdomain, which accounts for domain agnostic things like authentication, access control, auditing and so forth.

Of course in defining the architecture we need also to consider the setting that the final system will operate in, the “stakeholders” and their concerns, which is the subject of the next section.

2.3 Stakeholders and Concerns

There are many different actors (people, organizations, etc.) that have vested interest in the successful operation of a system for the personalized medicine. These can be organized in a “stakeholder onion diagram” [4] as the one shown in Figure 2 where we have following layers:

- The operational system. This identifies the stakeholder types that are in the vicinity of the system and its daily usage and includes its users (e.g. patients, clinicians, scientists, etc.), administrators responsible for the maintenance and the operational wellbeing of the system, and the system developers (software engineers, programmers, etc.). The concerns of the stakeholders in this layer are the functionality of the system, its usability, performance, and maintainability characteristics.
- The “containing system” groups together stakeholders that can have an effect in the p-medicine system from the point of view of the business, organization, or context that the system operates in. This context usually is the operational environment of a hospital, or the interaction with an external (interfacing) system and includes stakeholders such as the managers, the legal entities and other authorities. The primary concerns here are related to the patient safety, the security, the funding and cost of the operational system, and do on.
- The wider community, which can be citizens and organizations that became interested in the platform and want to learn more about it and experiment.

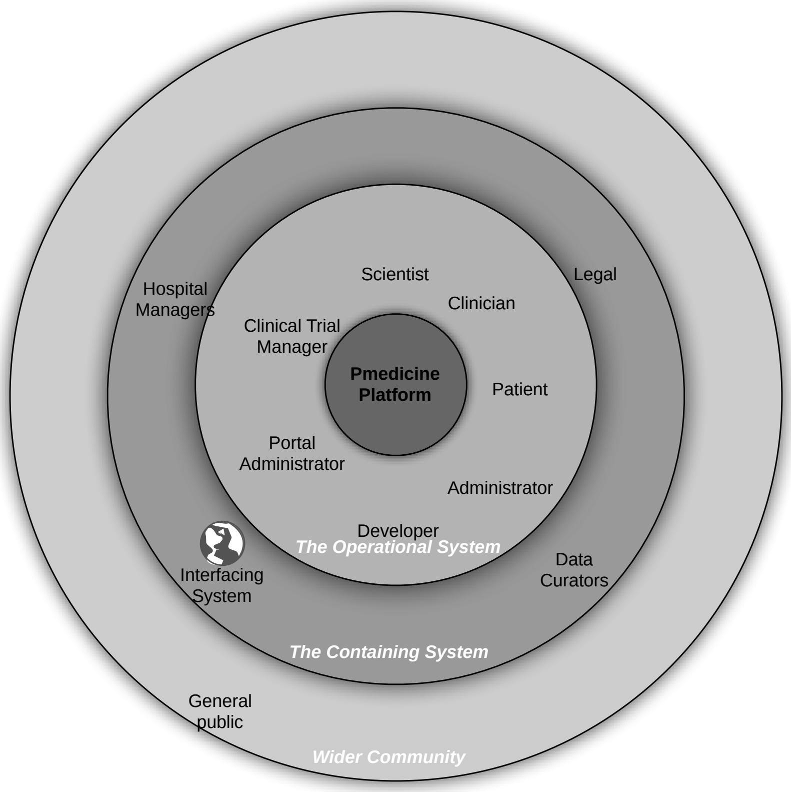


Figure 2 The p-medicine stakeholders layered

The concerns of the different stakeholder groups can be classified as follows:

- Patient data protection and privacy in the most important concern that is also the basis for other related concerns like the compliance to the EU legislation and the research ethics.
- Technical requirements and constraints are also important from the software engineering, administration, and maintenance points of view. Especially the use of "Cloud and high performance computing" as the "fabric" for the management and processing of data is nowadays introduced in almost every application domain and presents a lot of potential for personalized medicine as well.
- Usability requirements, friendly user interfaces that also adapt to the profile (e.g. patient, clinician) of the current user.

2.4 A layered view of the final architecture

It is customary to present a system architecture organized in layers. The layers organize related functionality and architectural concerns and usually are the following:

- Presentation: the layer of the end-user applications where the "user interface" / "human interface" components reside
- Application: The implementation of the core "business" logic and scenarios. This layer includes the main components that realize the perceived user functionality.
- The Domain or Data access and management layer, which is the layer that deals with the domain specific information and data.
- Infrastructure, where the database technologies, networking and other infrastructure elements reside. This layer is usually domain agnostic and generic.

The idea for having this layered architecture is that there's a more or less clear separation of responsibilities of the architectural components and that there's control over their dependencies because each layer directly interacts only with the layer beneath it.

Following this archetypical layered architecture style we can further partition, elaborate, and reposition each of the functional profiles and "bounded contexts" of the p-medicine system as shown in the following figure.

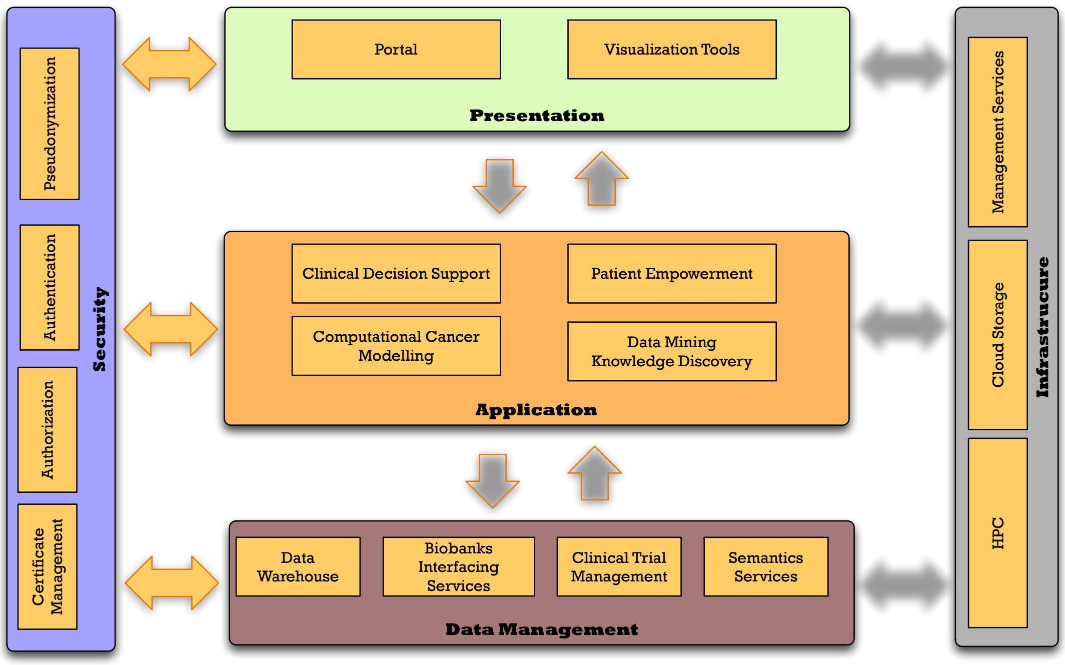


Figure 3 A layered view of the p-medicine architecture

As can be seen in Figure 3, we have introduced the security layer as a cross cutting concern because requirements such as the authentication and authorization of the users should be supported in every other layer in order to support the common security framework throughout the system. Another vertical layer is the infrastructure where the cloud storage and the high performance facilities are conceptually located. This is a domain agnostic layer of functionality but again pervasive in the sense that it supports all the other architectural elements. The Presentation layer is the user access layer that consists mainly of the p-medicine Portal and the visualization tools (e.g. image viewers and processing tools). The application layer is the core of the p-medicine system implementing most of the end-user scenarios (**Deliverable 2.2**) in the different application contexts. Finally the Data Management layer supports the Application and Presentation ones by providing the mechanisms for the efficient and secure upload, download, and management of data.

Nevertheless the layered presentation of the architecture described above hides important information about the p-medicine platform. There are certain operational and security related concerns that designate a more complex arrangement of the p-medicine components and the layered architecture of Figure 3 is too abstract to convey such finer details. Consequently, in Figure 4 we graphically present how the p-medicine components can be deployed into three different operational contexts or tiers that more clearly portray the context:

- The *clinical domain* is where the patient treatment takes place, for example in a hospital's premises. In this tier, users and affiliated personnel have full access to the patients' clinical records.
- The *clinical research domain* is where the clinical trial management happens. Here the patient data have gone through a first pseudonymization round but are still personal and the patients can be re-identified.
- The *personalized medicine research domain* is the core of the platform where truly anonymous data are stored. Data entering to this domain are gone through the second round of pseudonymization, are semantically annotated, and stored in the global data warehouse.

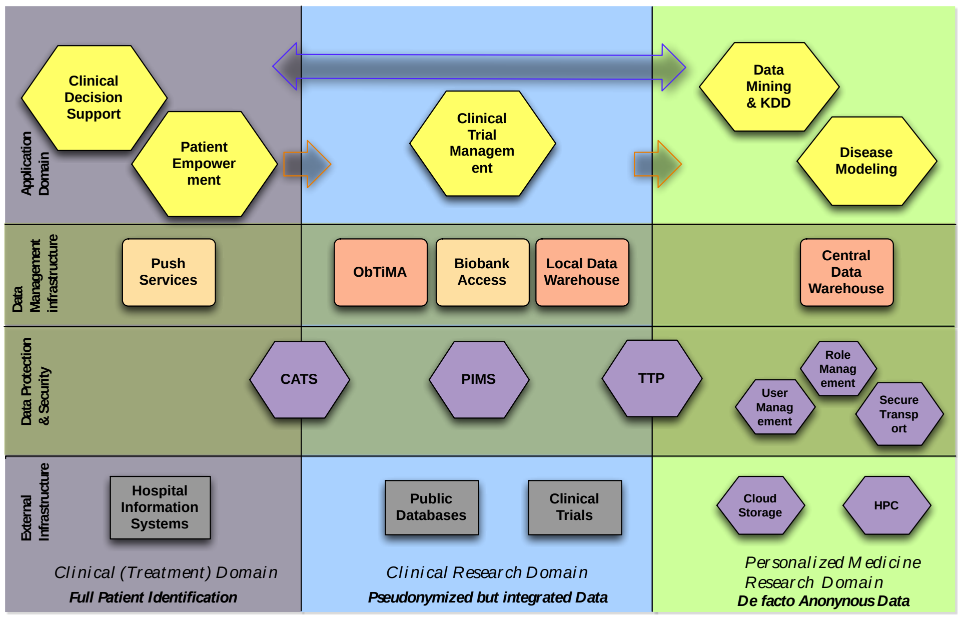


Figure 4 The p-medicine architecture divided in three operational contexts

This 3-tier data architecture also separates the p-medicine application areas so that for example patient empowerment is mostly related to the Clinical domain (because the patient has access to his own clinical record), while knowledge discovery and data mining is taking place in the core p-medicine research domain with the fully anonymous patient data. The p-medicine platform aims at providing tools, services, and technological solutions to all three operational contexts, as can be seen in the figure above.

3 The structure of the p-medicine system

In this section we follow the “domain driven design” and the separation of the p-medicine into different but interoperating domains that we described in Paragraph 2.2. Each of the identified “silos of functionality” (e.g. Clinical Decision Support, Computational Cancer Modelling, and so on) is a system of its own, with some particular requirements and specific architecture.

In each specific application domain we can identify a number of components or architectural elements that offer a certain set of functionalities. In the following sections we try to abstract the functionality of these components and present it in non-technical terms. The real interface using HTTP-based request response messages and specific message formats are described in subsequent sections.

Therefore for each component we provide:

- Its responsibilities, i.e. the description of what the component does
- Its collaborators, that is the components that this component interacts with
- Its (“abstract”) interface, which is the description of its operations, inputs, and outputs with no reference to specific technological and implementation details.

So in the following we are going to describe the architectural components from the point of view of the domain entities managed and the use cases they participate in, without delving into the software engineering details. For the description of the entities and their associations we will use the colour coding introduced by Coad [6] (please have look at Figure 5 for the explanation of the colours used). The technical details, such as the use of web services technologies for message delivery, will be detailed in Section 4 starting on page 29.

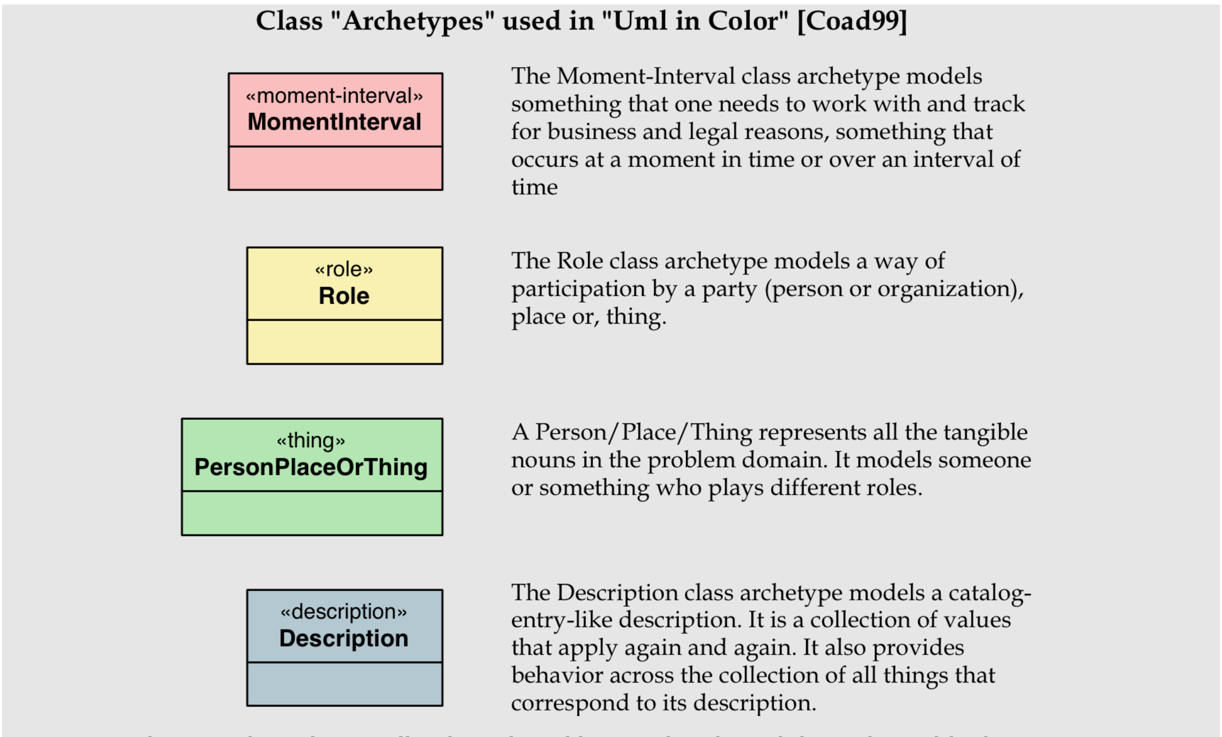

Class "Archetypes" used in "Uml in Color" [Coad99]
«moment-interval»
MomentInterval
The Moment-Interval class archetype models something that one needs to work with and track for business and legal reasons, something that occurs at a moment in time or over an interval of time
«role»
Role
The Role class archetype models a way of participation by a party (person or organization), place or, thing.
«thing»
PersonPlaceOrThing
A Person/Place/Thing represents all the tangible nouns in the problem domain. It models someone or something who plays different roles.
«description»
Description
The Description class archetype models a catalog-entry-like description. It is a collection of values that apply again and again. It also provides behavior across the collection of all things that correspond to its description.

Figure 5 The colour coding introduced by Coad and used throughout this document

The "Hexagonal Architecture"

In describing the architecture of the specific application domains of the p-medicine platform we could follow the same layered approach we showed in Figure 3. The layered approach is of course a conceptual framework. In reality and in many cases there still dependencies among components in non-adjacent layers (transitive dependencies are still dependencies) and there's a lot of coupling on low level details (e.g. the choice of frameworks, databases, etc.).

There are alternative approaches in designing applications that put the layers in a different perspective. Figure 6 shows one of these alternative views and it's strongly influenced by the "hexagonal architecture" (or the "ports-and-adapters") by Alistair Cockburn [5]. The primary difference of this new perspective is that the domain and application layers do not depend on the other layers, the presentation (UI) and the infrastructure. Instead, the entire coupling is towards the centre, i.e. the code depends on layers more central while the domain layer uses the external "rings" through some generic abstractions (interfaces, adapters). The domain layer is at the heart of the system, it's the most important layer since it models the concepts, ideas and rules within the software that are left after stripping away the stuff specific to the technologies we are using.

This different perspective can be used in describing the architecture of an application by prescribing a certain methodology: starting from the domain model (i.e. the data and the information managed by the system to be designed) and the application layers while postponing the selection of the infrastructure (databases, communication mechanisms, etc.) and the design of the user access mechanisms to a second stage.

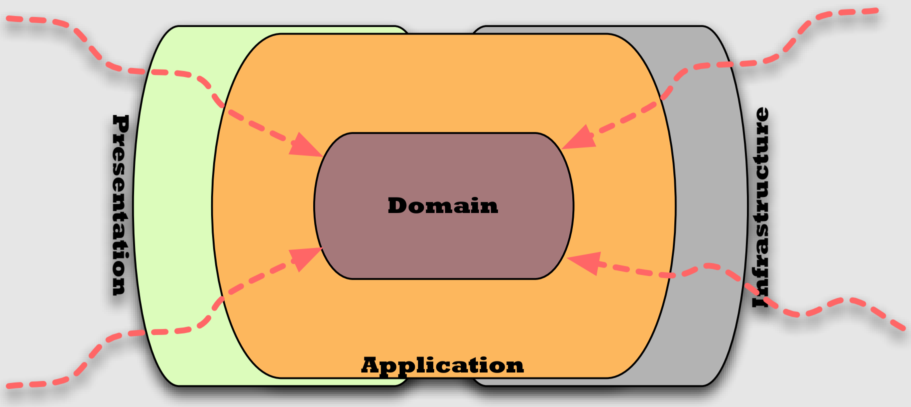


Figure 6 A domain centred view of the architecture

3.1 Authentication and Authorization

The p-medicine security framework is designed around the Security Assertion Markup Language (SAML) standard and a number of WS-Security specifications. A p-medicine Identity Provider (IdP) provides identity assertions to all services within p-medicine, effectively providing single-sign-on (SSO) capabilities to the whole platform.

Web sites integrate with the IdP by providing a SAML compliant Identity Consumer (see ***Deliverable 3.4*** *"Service Integration Guidelines"* for more details). An Identity Consumer consumes and validates the assertions provided by the IdP. For non user-interfacing clients such as REST Web Services, a SAML Identity Assertion is transmitted through the HTTP Authorization header (please see again *Deliverable 3.4* for more details).

On the authorization front, the privacy and security concerns require the introduction of access control rules for all the data managed by the p-medicine system. These requirements led to the introduction of policy-based authorization services build upon the eXtensible Access Control Markup (XACML) Language. In XACML, an access request is modeled as a “subject” who wants to perform an “action” on a “resource” (subject/action/resource triplet) and its underlying model can be seen in Figure 7. More details about the authorization framework and its various services and roles, like policy enforcement points (PEP), policy decision points (PDP) and policy administration points (PAP), can be found in **Deliverable 3.4**.

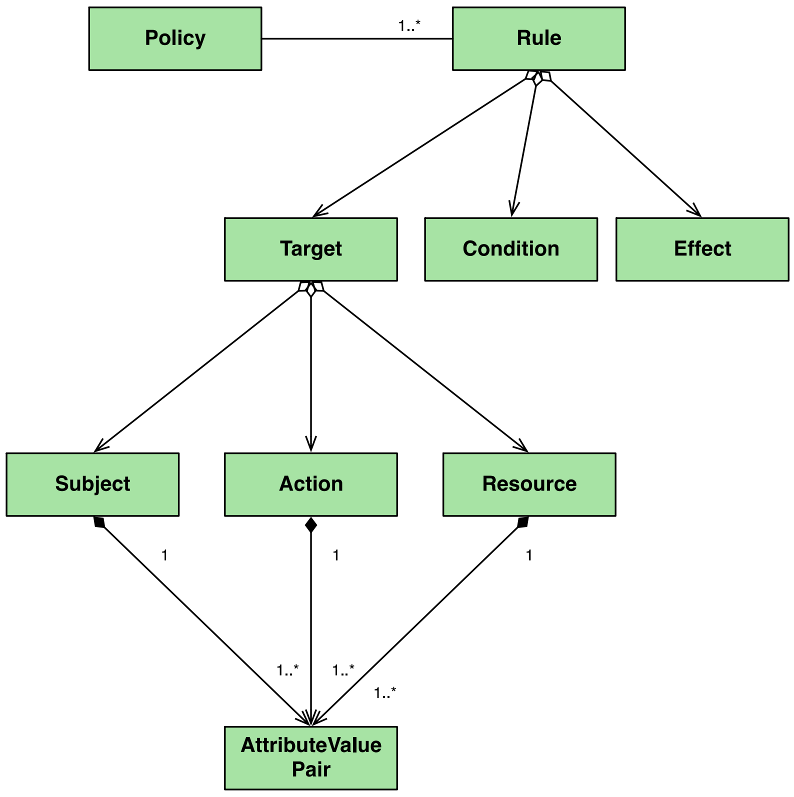


Figure 7 The XACML based Attribute Based Access Control model

The User Management is the other major component of the security framework which is responsible for user enrolment, user identity and credential management. Its underlying domain model is shown in the next figure.

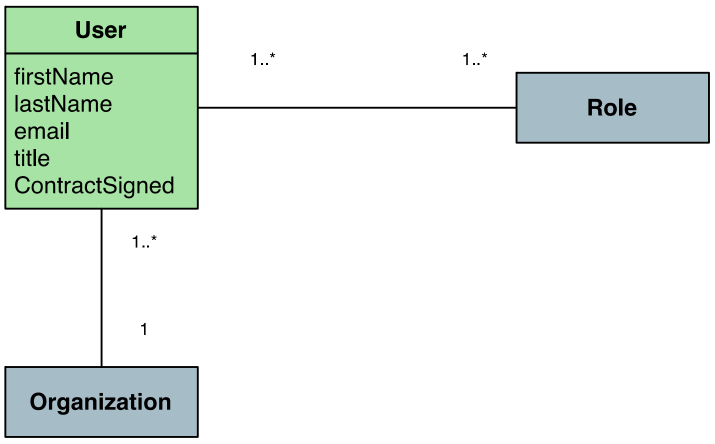


Figure 8 The users management domain model

For each user there's basic profile information like the name, email, and title and an indication whether he/she has signed the data sharing contracts of the p-medicine system. Finally, information about the roles the user has and the organization he/she is affiliated with are also registered and subject to the attribute based access control of the authorization infrastructure.

3.2 Data Management

The management of the data is the most important functional context because the data have a central role in a personalized medicine environment. We are primarily interested in the handling of patient data, which means that the primary concerns are the privacy and the conformance to the ethical and legal guidelines and requirements. **Deliverable 5.1** "Setting up of the data protection and data security framework" includes a brief overview of the legal rules concerning the use of personal data on a European level.

Based on the data protection requirements the data architecture of the p-medicine introduces a clean separation of the *clinical domain* and the *research domain*. Within the research domain only de facto anonymous data shall be used. Therefore, a data protection framework based on a double pseudonymisation procedure has been devised with the Center for Data Protection (CDP) as a central data protection authority and a Trusted Third Party (TTP) as a trusted data custodian. Thus the p-medicine domain is a network of trust based on contractual agreements that shall ensure the compliance to the data protection rules set up for the framework (Deliverable 5.1).

In Figure 9 below we see the process of uploading of new data into the p-medicine platform as they move from the clinical domain (e.g. a Hospital) where they are pseudonymized, to the TTP zone where the second anonymization takes place, and finally to the p-medicine data warehouse.

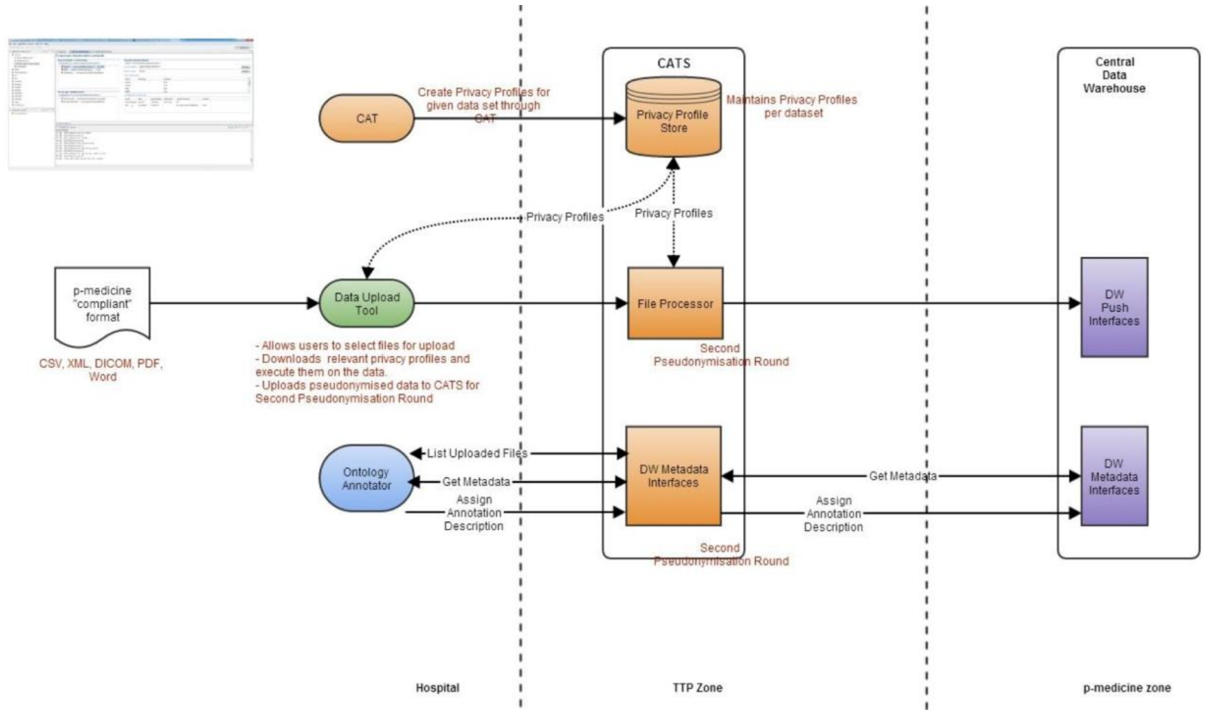

CATS
Central Data Warehouse
CAT
Create Privacy Profiles for given data set through CAT
Privacy Profile Store
Maintains Privacy Profiles per dataset
Privacy Profiles
Privacy Profiles
p-medicine "compliant" format
CSV, XML, DICOM, PDF, Word
Data Upload Tool
File Processor
DW Push Interfaces
- Allows users to select files for upload
- Downloads relevant privacy profiles and execute them on the data.
- Uploads pseudonymised data to CATS for Second Pseudonymisation Round
Second Pseudonymisation Round
Ontology Annotator
List Uploaded Files
Get Metadata
Assign Annotation Description
DW Metadata Interfaces
Get Metadata
Assign Annotation Description
DW Metadata Interfaces
Second Pseudonymisation Round
Hospital
TTP Zone
p-medicine zone

Figure 9 The anonymization and ontology annotation steps in the data upload process

3.2.1 Initial Upload and the Pseudonymization framework

Before the data enter the p-medicine research area they need to be anonymized. The details of the double anonymization procedure and the related software components are described in **Deliverable 8.3**.

As shown in Figure 9 above the data need to go through the Trusted Third Party (TTP) before they are stored in the data warehouse. The end user application for the initial upload of the data to the TTP is the Data Upload Tool that is described in **Deliverable 8.5**. For a given user database or file there should be a “privacy profile” that is a set of transformation rules for the pseudonymisation of patient identifying data. The privacy profiles are created with the Custodix Anonymization Tool (CAT) and then uploaded to the Custodix Anonymisation Tool Services (CATS) server. The Data Upload Tool uses the Java library of the CATS tool and an existing privacy profile to perform locally, in the clinical domain, the first round of pseudonymization. An additional component used in this use case is the Patient Identity Management System (PIMS) that supports unique identification of a patient in different administrative domains by assigning pseudonyms. Please see **Deliverable 8.3** for more details on these components and their interactions.

After the first pseudonymization round, the (pseudonymized) data are uploaded into the TTP. There, the data need to go under the scrutiny of CDP and the second round of pseudonymization takes place that results in new pseudonyms by the transformation (using a cryptographic operation) of the original pseudonyms. Finally the now *de facto* anonymous data are uploaded from the TTP to the p-medicine data warehouse.

3.2.2 Ontology Annotation and Translation

After the data enter the p-medicine zone and stored in the filestore of the data warehouse the ontology annotation and translation takes place (Figure 10).

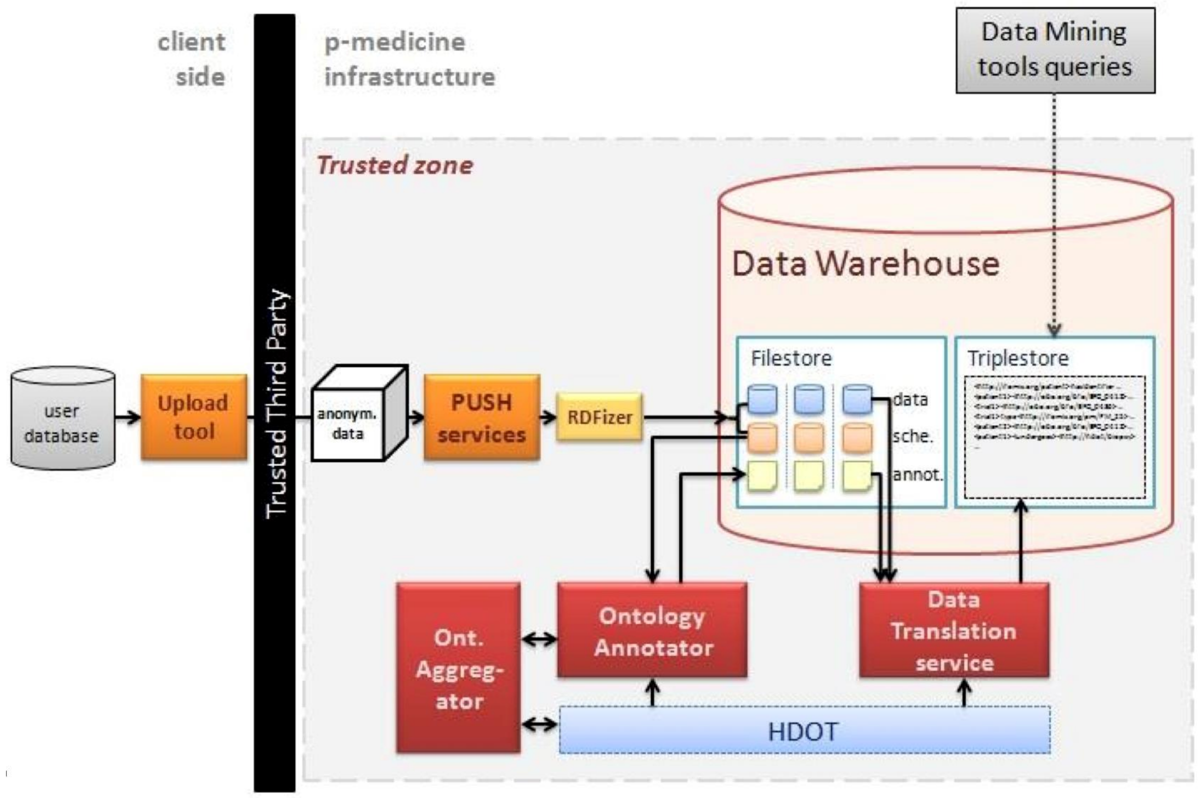


Figure 10 The ontology annotation and translation processes that take place in the p-medicine zone

This step is needed because the Data Warehouse is also backed up by an RDF “TripleStore” i.e. the uploaded databases and files are transformed to Semantic Web-compliant “triples” and semantically annotated using p-medicine’s Health Data Ontology Trunk (HDOT) ontology. This is a requirement described in the **Deliverable 4.2** where the p-medicine semantic layer is discussed. HDOT itself and the p-medicine ontology suite are presented in **Deliverable 4.1**.

The ontology annotation step is required in order to provide a mapping from the anonymized data to the HDOT concepts and relationships. The data are transformed to RDF by some file-type specific “RDFizer” and then the user (either the original uploader or an ontology expert) launches the Ontology Annotator to create the mapping from the “local” schema of the uploaded data to the global schema of HDOT. Please keep in mind that this step is not required if the data have been already annotated with HDOT terms as is the case with the ObTiMA data.

The Data Translator service is then put in charge of performing the actual translation of data from the annotated databases to an HDOT compliant format. The Data Translator takes as input the data of one database and its ontological annotation (previously generated with the Ontology Annotator or ObTiMA), and returns the data translated to an HDOT compliant form. The new triples are then inserted into the Data Warehouse’s triplestore.

The Ontology Annotator and Data Translator tools are described in **Deliverable 4.3**.

3.2.3 Data Warehouse

Responsibilities

The data warehouse is the responsible for the storage and management of different types of medical data. It can manage databases in CSV and Microsoft Access formats, clinical files, and also manages DICOM images access. The most interesting point is that data warehouse extracts semantic information from all of the types, storing it as RDF triples in the triplestore. This information then, is automatically ready to be queried by authorised third-parties.

The definition of this component alongside its requirements is available in **Deliverable 7.1** (“*Report on overall design including VPH-Share 2.2 and indicating its impact*”). A depiction of the main entities managed by the data warehouse is shown in Figure 11 using the colour coding of Peter Coad in [6].

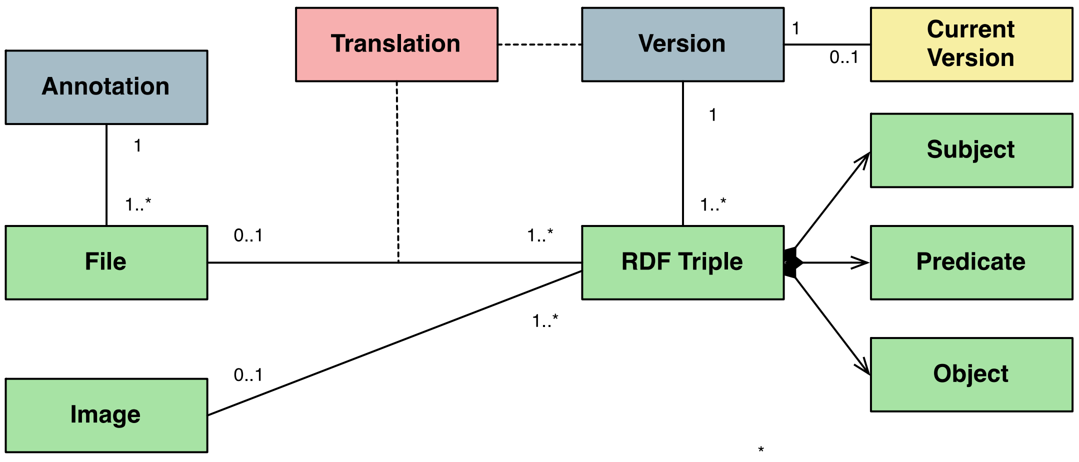


Figure 11 The "domain" of the Data Warehouse

The main domain entities managed by the data warehouse are:

- Generic files will be stored in the data warehouse. Files in the store will be referred to by URI, since many federated file stores may exist and they may be referred to by the structured data in other data warehouses. There is no specific need to keep any information about the file beyond its name, and the content of the file. File metadata

and relationships between files (resembling a hierarchy) are stored in the structured data store (i.e. the triplestore).
- DICOM Images: DICOM (Digital Imaging and Communications in Medicine) is the de facto standard for handling, storing, printing, and transmitting information in medical imaging. The image store should offer direct, secure access to images through the standard DICOM image access protocols.
- Structured Data: The core data that will be stored in the warehouse is structured data. Since ontologies play a key part in the p-medicine project, the structured data should be transformed into the HDOT format to be saved at the data warehouse.

Collaborators

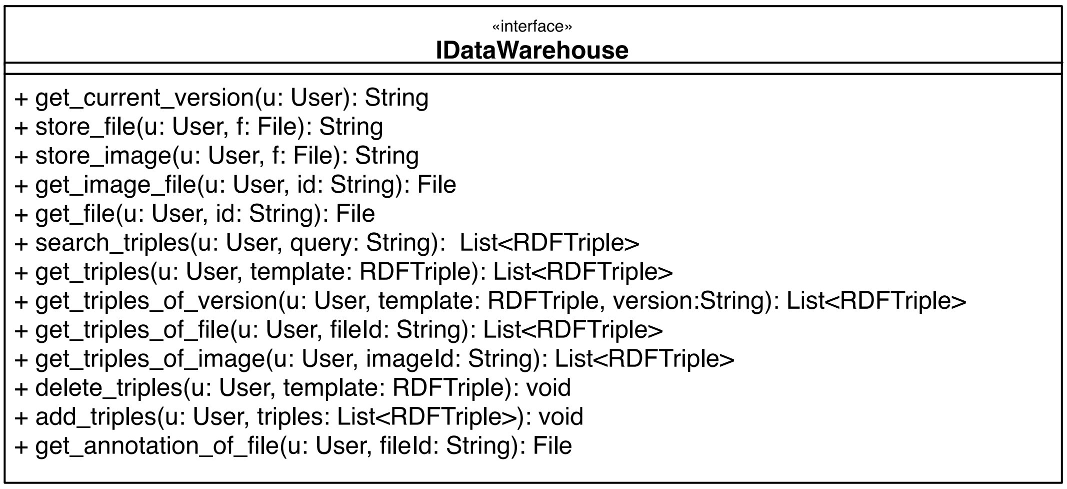
The user does not directly interact with it: the data warehouse is intended to be used by other components from the p-medicine project as a central storage service. The components that directly interact with the data warehouse are the Data Translator tool, which is triggered every time a new database is added to the data warehouse, the Security Framework, which manages the permissions on accessing the data warehouse, the TTP services, which insert new elements in the data warehouse, the ObTiMA front-end system, which is the user-interface tool for the data in the data warehouse, and the images server, which provides DICOM images access for the data warehouse. Furthermore the data warehouse makes use of the cloud storage services for its file store.

Interface

The Data Warehouse offers the following functionality (Figure 12):

- Get current version of the data warehouse contents: **`get_current_version(u: User): String`**
- Upload a new file: **`store_file(u: User, f: File): String`** The file is stored in the Filestore and its identifier is returned
- Upload a new image file: **`store_image(u: User, f: File): String`** The file is stored in the Imagestore and its identifier is returned
- Retrieve Image Files from ImageStore given their identifier: **`get_image_file(u: User, id: String)`**
- Retrieve Files from Filestore given their identifier: **`get_file(u: User, id: String)`**
- Search triplestore given a SPARQL query: **`search_triples(u: User, query: String)`**
- Search triplestore using “query-by-example”: **`get_triples(u: User, template: RDFTriple)`**
- Search triplestore using “query-by-example” in a specific version of the triplestore: **`get_triples_of_version(u: User, template: RDFTriple, version: String)`**
- Get the triples that were produced during the translation of a specific file: **`get_triples_of_file(u: User, fileId: String)`**
- Get the anotation (mapping file) of a specific file: **`get_anotation_of_file(u: User, fileId: String)`**
- Add triples to the triplestore: **`add_triples(u: User, triples: List<RDFTriple>)`**
- Delete triples from the triplestore: d**`elete_triples(u: User, triples: List<RDFTriple>)`** (After the deletion of the file the associated RDF triples are also removed from the triplestore).

The precondition for invoking these operations is that the invoker has been authenticated as a legitimate p-medicine user and therefore provides the relevant **`User`** information.



«interface»
IDataWarehouse

```
+ get_current_version(u: User): String
+ store_file(u: User, f: File): String
+ store_image(u: User, f: File): String
+ get_image_file(u: User, id: String): File
+ get_file(u: User, id: String): File
+ search_triples(u: User, query: String):  List<RDFTriple>
+ get_triples(u: User, template: RDFTriple): List<RDFTriple>
+ get_triples_of_version(u: User, template: RDFTriple, version:String): List<RDFTriple>
+ get_triples_of_file(u: User, fileId: String): List<RDFTriple>
+ get_triples_of_image(u: User, imageId: String): List<RDFTriple>
+ delete_triples(u: User, template: RDFTriple): void
+ add_triples(u: User, triples: List<RDFTriple>): void
+ get_annotation_of_file(u: User, fileId: String): File
```

Figure 12 The abstract interface of the Data Warehouse

3.3 Computational Cancer Modelling

Deliverable 12.1 describes the p-medicine simulation scenarios for breast cancer, Wilm's tumour, and Acute Lymphoblastic Leukemia. This application area has a strong need for computational power, more than the other p-medicine application domains. Therefore a major concern is the use of high performance (HPC) facilities, such as clusters of machines and the computational Cloud.

The main domain entity here is the computational model, which is a "mathematical model implemented in a computer system that requires high performance computational resources to execute" [7]. The computational needs for the cancer modeling have been demonstrated and addressed as described in **Deliverable 12.3** "*Report on the Development of the Oncosimulator and the Utilization of the Biomechanism Models*" through the use of Graphics Processing Units (GPU). Furthermore, with the cooperation of VPH-Share consortium the p-medicine computational models have been demonstrated to run on the VPH-Share cloud infrastructure.

A model of this kind incorporates simulations of cellular, molecular, and other processes in living organisms and requires input data and parameters values for its initialization while it (usually) produces some visualization (e.g. for tumour growth) and output values. The execution of a Model given some inputs data and parameters creates a "simulation", a domain entity encapsulating the specific run and its results (Figure 13). During this run the simulation passes through a series of statuses (e.g. starting, running, finished, or failed).

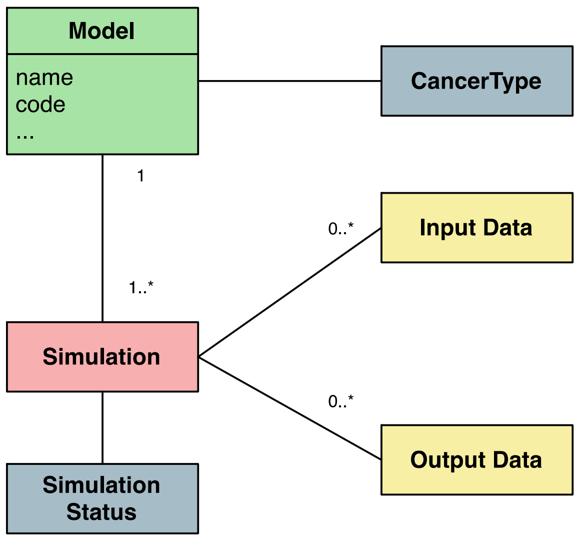


Figure 13 The computational modelling domain

To support the execution and the monitoring of the models' execution and to abstract over the differences between the execution frameworks (e.g. GPUs versus Cloud) a Simulation Service interface can be defined:

«interface»
ISimulationService

+ start_simulation(u: User, m:Model, input: List<Data>): String
+ get_simulation_status(u: User, s: String): Enum
+ get_results(u: User, s:String): List<Data>

Figure 14 The Simulation Service abstract interface

The operations supported by this abstract interface are:

- Start a new simulation: **`start_simulation(u: User, m: Model, input: List<Data>): String`** The caller of this operation should provide the model to run and a list with the required input data. The return value is the identification of the simulation (run) just triggered.
- Query the status of a simulation, given its identification: **`get_simulation_status(u: User, s: String): Enum`**
- Retrieve the results of a simulation as a list of output data given its identification: **`get_results(u: User, s: String): List<Data>`**

3.4 Patient Empowerment

The goal of the Interactive Empowerment Services system is to help the patients to understand the medical documentation and to help them make informed choices. To this end in this application domain of personalized medicine, the following components are introduced:

- The ALGA-C questionnaire which provide clinicians with information about the cognitive attributes of each individual patient
- The Personal Health Record (PHR), which is used by the patients to overview and, in some cases, update their clinical record, i.e. their drugs, chronic and other diseases, etc.
- Domain specific support tools, such as components for validating the medication list of the patient and reporting drug-drug interactions.

- Informed Consent management tools such as the “Donor’s Decision Tool” (**Deliverable 14.3**) that allow patients to be the owners and managers of their data.

3.4.1 The ALGA-C questionnaire

The ALGA-C questionnaire provides a psycho-cognitive picture of the patient whom the physician is about to meet, so that he can optimize the communication style and catalyse the interaction (see deliverables **Deliverable 14.1** and **Deliverable 15.4** for more details respectively on development and validation of the two versions of ALGA questionnaire, for healthy people and cancer patients). As shown in Figure 15 below a patient can fill-in the ALGA-C Questionnaire multiple times, before seeing their doctor or at any other time through their PHR application, and the answers to the questionnaire are used to build the psycho-cognitive profile of the patient. For the result profile a number of recommendations can be given to the physician in order to adapt his style of communication and improve the interview or visit.

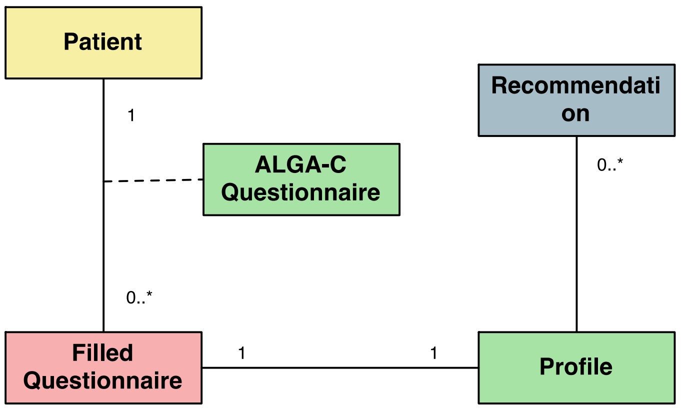


Figure 15 The ALGA-C Questionnaire model

The questionnaire requires a user-interfacing tool for the patient to fill it. The management of the questionnaires is done centrally by a “Profiling service” that allows accessing the results, patient profiles and recommendations, as shown in Figure 16.

«interface»
IQuestionnaireProfiler

+ get_questionnaires(u: User, p: Patient): List<Questionnaire>
+ get_recent_questionnaire(u: User, p: Patient): Questionnaire
+ get_profile(u: User, q: Questionnaire): Profile
+ get_recommendations(u:User, p: Profile): List<Recommendation>

Figure 16 The ALGA-C Profiling abstract interface

Details for the implementation of the questionnaire application and the profiler service can be found in **Deliverable 14.4**.

3.4.2 Personal Health Record

The PHR is the patient managed electronic health record. It is an external application in the vicinity of the patient but in any case outside the p-medicine domain. The consortium has selected IndivoX[3] as the specific implementation of the PHR because it allows for extensibility. Therefore we have extended its user interface and functionality by incorporating the ALGA-C questionnaire for the patient to fill it and also by taking advantage of its results (coming from the Profiling service) to adapt PHR’s interface based on the most recently

[3] http://indivohealth.org

extracted psycho-cognitive profile of the patient. Description of the possible adaptations is given in **Deliverable 14.4**.

3.4.3 Patient Empowerment support services: Drug-Drug interactions

The Drug-Drug interaction service is an example of a supporting tool that can be used by the physicians and the patients alike. Its aim is to check whether two or more drugs can be safely used together. This is an important functionality when a patient through his/her PHR updates the medication list with a newly (possibly un-prescribed) medicine e.g. an antibiotic.

The implementation uses the publicly available DrugBank [8] database.

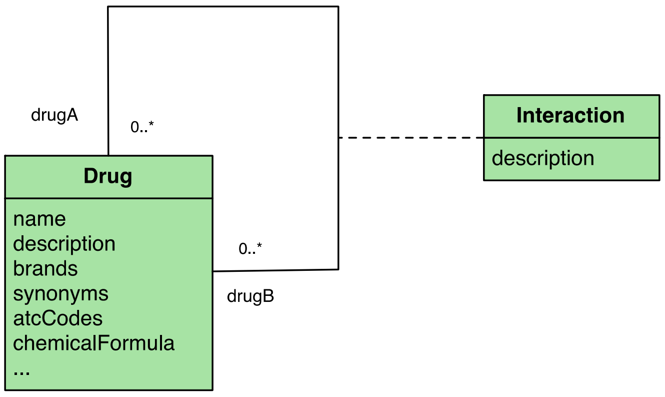


Figure 17 The drugs and their interactions

This simple tool allows its collaborators (mainly the PHR but also any other component of the p-medicine system) to retrieve the interactions of a single drug or the possible interactions of a set of drugs, as shown in Figure 18. Please note that this service does not require any authentication information in order to be used.

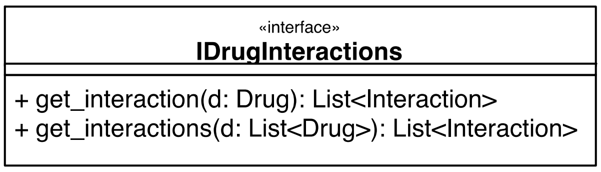

«interface»
IDrugInteractions
+ get_interaction(d: Drug): List<Interaction>
+ get_interactions(d: List<Drug>): List<Interaction>

Figure 18 The abstract interface of the Drug Interaction service

3.4.4 Consent Services

The patient's written Informed Consent is a mandatory prerequisite for any type of biomedical research. **Deliverable 14.3** describes the tools for the integration of the ethical-legal information on consent given by donors into biobank data.

The underlying domain model for the informed consent deals with the management of the patients' consent and the relevant data. More specifically, as shown in Figure 19, the consent of a patient is provided in the context of a «project» that can be a clinical trial or other research study. In other words, the «project» represents the extent for the informed consent and therefore the scope which the patient's biomaterial can be used for. The consent is given at an instant in time and possibly is held for a time period i.e. until it is withdrawn. Associated with a «project» there are the consent document itself and possibly other descriptive information that provide additional documentation and explanations (e.g. multi-media content regarding the study to enhance comprehension and retention).

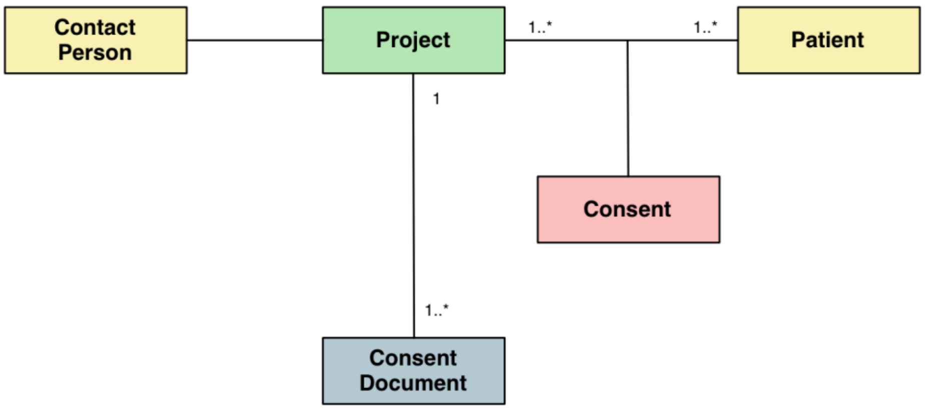


Figure 19 The main entities in the e-Consent context

Figure 20 shows the overall architecture for the provision or withdrawal of the patient's e-consent and how this affects the management of their clinical data. The primary stakeholders in this domain are the following:

- The treating *physicians* that provide a short summary of the project in layman's language and in the mother tongue of the patient. This needs to explain the reason why the biomaterial is needed and what will be the expected result.
- The *patients* that provide or withdraw their consent for the sharing and the use for research purposes of their biomaterial and related data.
- The *researchers* that have access to the material the patients have provide consent for.

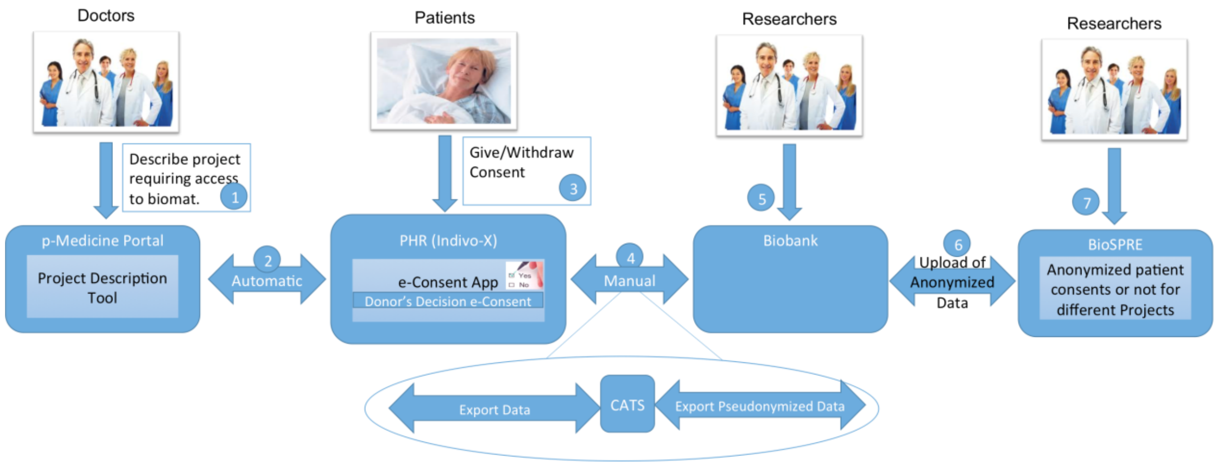

Doctors
Patients
Researchers
Researchers
Describe project requiring access to biomat.
1
Give/Withdraw Consent
3
5
7
p-Medicine Portal
Project Description Tool
2
Automatic
PHR (Indivo-X)
e-Consent App
Donor's Decision e-Consent
4
Manual
Biobank
6
Upload of Anonymized Data
BioSPRE
Anonymized patient consents or not for different Projects
Export Data
CATS
Export Pseudonymized Data

Figure 20 Overview of the interactions and the components in the e-Consent services

In terms of the technical architecture the following components operate in this context:

- The *Portal* is the tool used by the physicians for describing the terms of the consent and managing the relevant documents. All the consent related information is stored in a portal accessible and authorized database.
- The *Personal Health Record* (PHR) portal and specifically its e-Consent application present the consent forms and related documents to the patients. A patient is then supported through this application to provide their consent or even withdraw it at a later time, while at any time can review his/her consents and their terms.
- The *Biobank platform* comprises the relevant information system and the Biobank access framework (p-BioSPRE) through which accessing biobanks and sharing biomaterial is greatly simplified.

- The communication between the PHR and the Portal, shown as number (2) in Figure 20, is supported through a programmatic interface that allows accessing the details of the available “projects” and their consent documents and supporting documentation. This interface shown in Figure 21 is provided by the Portal, or actually a component hosted together with the Portal and having access to the same consent/”project” database, and it is invoked by the PHR in order to present the consent related information to the patients.

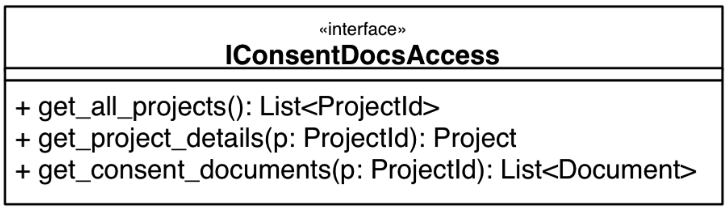

«interface»
IConsentDocsAccess
+ get_all_projects(): List<ProjectId>
+ get_project_details(p: ProjectId): Project
+ get_consent_documents(p: ProjectId): List<Document>

Figure 21 The consent documents access interface

As can be seen in Figure 20 the consent related documents, which are prepared by the ethical and legal committees and enhanced by healthcare professionals, are uploaded into the p-medicine portal while the actual capture of the patient’s consent takes place in the PHR. After the patient’s consent has been given, their biomaterial can be shared in the context of the associated “project” but the rest of this scenario requires to a large extent manual interaction. A major complication is that biobanks and the rest of the infrastructure deals with anonymised data and therefore the donor’s real identifier, used in the context of PHR and informed consent, should be associated with the corresponding pseudonymised samples held in the biobanks. This process is greatly facilitated by the use of the CATS and related security and pseudonymisation infrastructure but nevertheless cannot be totally automated, in the same vein as the general data upload and pseudonymisation process can not be fully computerized (see paragraph 3.2.1). The same process should be followed in the case donors withdraw their consent, where the associated samples must be located and destroyed.

3.5 Clinical Decision Support

A Clinical Decision Support System (CDSS or CDS) is an interactive computer software system designed to assist physicians and other health professionals with decision-making tasks, as determining diagnosis of patient data. CDS systems by definition are in the clinical domain that is outside of the research domain where the core of the p-medicine system is located. Nevertheless, the CDS application domain is in the fringes of the p-medicine system by taking advantage of the p-medicine managed data and the knowledge extracted from these data.

Deliverable 13.1 presents the requirements for the clinical decision support tool and also describes a number of scenarios where data mining tools (e.g. literature mining for the identification of Severe Adverse Events - SAEs) and computational models like the oncosimulator to predict the likely response of a given patient’s breast cancer. **Deliverable 13.4** presents the details of the CDS prototype and its generic architecture (Figure 22).

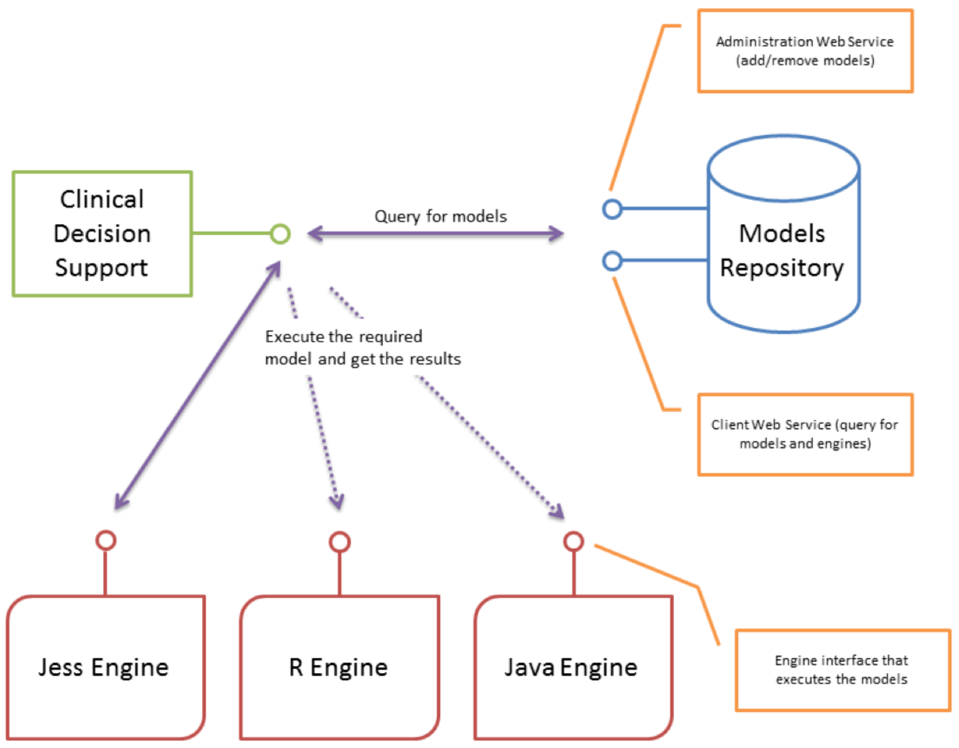

Administration Web Service (add/remove models)
Clinical Decision Support
Query for models
Models Repository
Execute the required model and get the results
Client Web Service (query for models and engines)
Jess Engine
R Engine
Java Engine
Engine interface that executes the models

Figure 22 The CDS Framework 's architecture

The integration of the p-medicine computational models with the CDS system was further described in **Deliverable 13.7**.

3.5.1 The p-medicine workbench

This tool is essentially a "tool registry" where the p-medicine tools and other tools outside of p-medicine are registered. The annotation of tools includes a classification hierarchy borrowed from VPH Toolkit, a semantic based classification using EDAM[4] terms, and user-submitted "tags" (keywords). The users can submit free text queries that match tools' names and descriptions or keywords, semantic terms to filter the data that the tools produce or consume or their functionality, etc. There's also ongoing work for supporting natural language processing of user submitted queries.

The user does not directly interact with the Workbench. Instead, there's the workbench "portlet" in the p-medicine portal that provides the user interface for this service. So far this portlet is its only "client" but the interface is open for all the p-medicine functional components to use it.

[4] http://edamontology.org/

The generic application architecture is shown in the right (Figure 23). Generally there can be a number of p-medicine tools and services that the Workbench server contacts. In principle the workbench server should also contact other tool registries such as the VPH Toolkit but so far this has not been implemented (and actually the VPH Toolkit lacks any API for automating this interaction.)

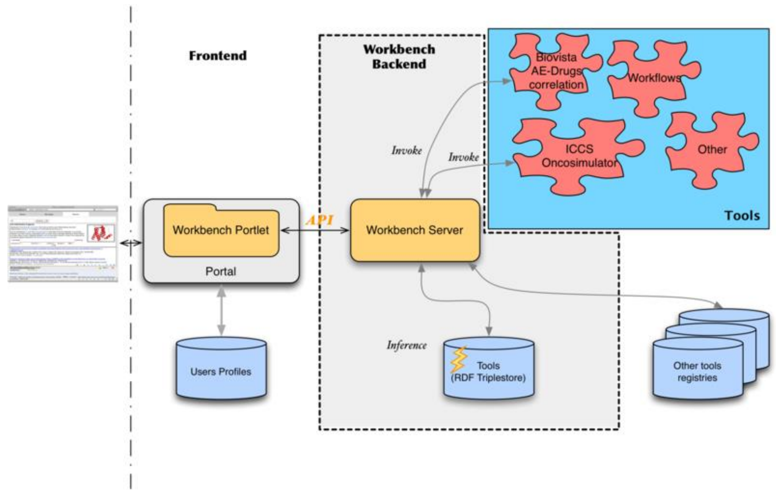


Figure 23 The architecture of the p-medicine workbench

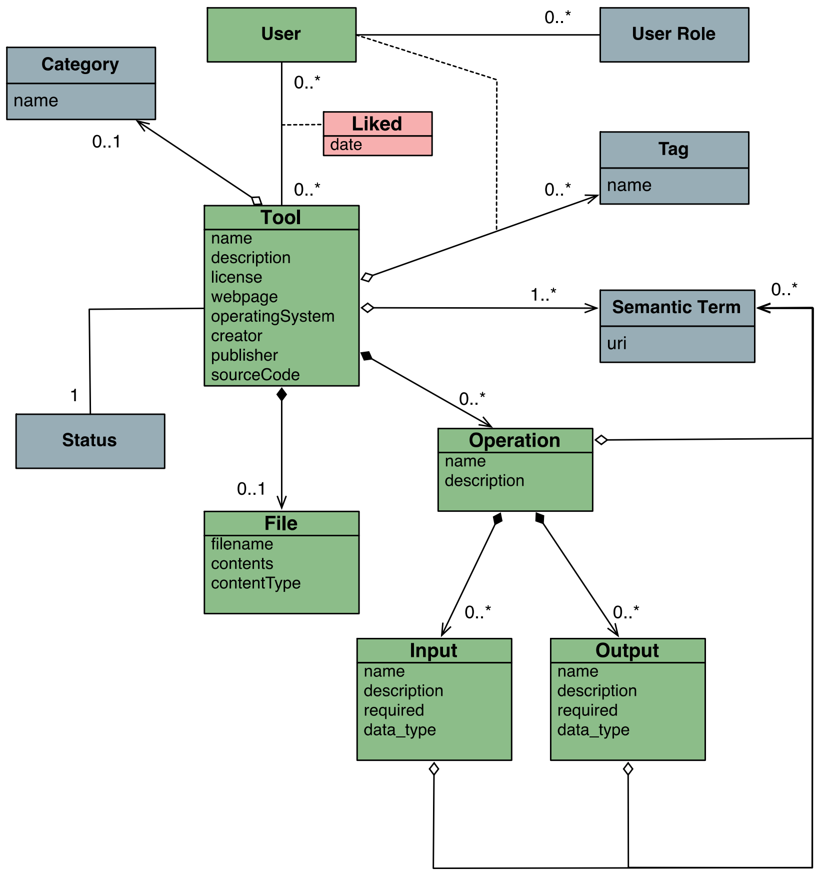
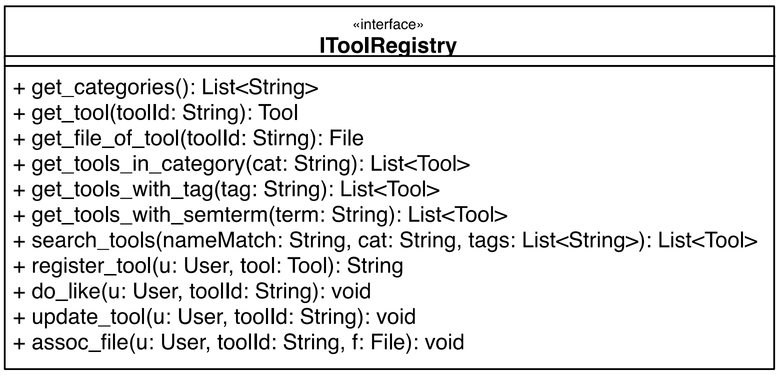


Figure 24 The domain model of the p-medicine workbench

Its “domain model” includes the Tool as the central domain entity that can have a number of operations, each with a set of inputs and outputs (Figure 24). Each tool, operation, input, and output can be annotated with semantic terms from the EDAM ontology while Tools can also be classified into a predetermined (static) hierarchy of categories and in an open ended, unstructured set of Tags (keywords) submitted by the users. The users can also register their approval for specific tools using the “like” functionality of the system. The interface of the p-medicine workbench is shown in Figure 25.



```
«interface»
IToolRegistry

+ get_categories(): List<String>
+ get_tool(toolId: String): Tool
+ get_file_of_tool(toolId: Stirng): File
+ get_tools_in_category(cat: String): List<Tool>
+ get_tools_with_tag(tag: String): List<Tool>
+ get_tools_with_semterm(term: String): List<Tool>
+ search_tools(nameMatch: String, cat: String, tags: List<String>): List<Tool>
+ register_tool(u: User, tool: Tool): String
+ do_like(u: User, toolId: String): void
+ update_tool(u: User, toolId: String): void
+ assoc_file(u: User, toolId: String, f: File): void
```

Figure 25 The interface of the workbench

4 Technical description of the Components

The REST architectural style has been selected for the implementation of the platform. An introduction to this style and its adaption to the HTTP and web services related technologies can be found in Chapter 6 in the **Deliverable 8.6.1** "Integration guidelines and monitoring of tools and services".

REST is resource oriented or data-oriented rather than service oriented. In the HTTP realization of the REST style, there is a predetermined set of operations (HTTP methods like GET and POST, the "verbs") and a potentially infinite, application specific set of resources (e.g. "questionnaire", the "nouns") Therefore, in the following sections we see how the abstract interface that is method and operation centred is transformed to be resource (data) centred.

Most of the components are implemented as HTTPS/REST based services. There are some notable exceptions:

- The communication with the security infrastructure, such as the Identity Provider (IdP), uses the SOAP/WSDL-based WS-Security (e.g. WS-Trust). But the transmission of the SAML tokens is implemented with the traditional HTTP "Authorization" header, which makes secure component interactions RESTful again.
- Some components like the Data Translator and the Custodix Anonymization Tool (CAT) are provided as Java libraries for performance reasons, i.e. all communication is local and in memory. Nevertheless we still consider them to be components since they offer clearly specified functionalities.

4.1 Data Warehouse

The Data Warehouse component provides data storage and it can be accessed through a RESTful API.

Accessing Triples

HTTP Method	**`GET/POST`**
HTTP Endpoint	**`/v1/triplestore`**
Query Parameters	**`query`** : URL-encoded SPARQL query **`timeout:`** Optional limit in whole seconds on evaluation time of query. If omitted, query time is unlimited
Formats	Comma separated values responses for SELECT queries with the binding variable as the column names in the first row, followed by a row per binding result
SAML Authorization	Yes
Description	This is a resource for accessing triples in the main data warehouse triplestore. This interface is based upon the OpenRDF Sesame RESTful interface. It supports both GET (preferred) and POST that is useful when the query is considered too long to be transmitted as part of the request URL. In the case of POST the query parameters MUST be passed in the request body as web form data as per HTML 4.01 section 17.13.413

Accessing Version

HTTP Method	**`GET`**
HTTP Endpoint	**`/v1/triplestore/version`**
Query Parameters	None
Formats	None
SAML Authorization	Yes
Description	Returns the current version of the triplestore and date and time of last change in the response headers: • X-Version: Integer version number of the triplestore. • Last-Modified: HTTP-date of the last change to the triplestore, when this version was created.

Statements Resources

HTTP Method	**`GET/POST/PUT/DELETE`**
HTTP Endpoint	**`/v1/triplestore/statements`**
Query Parameters	GET • **`subj`**: Optional subject which all returned statements should have • **`pred`**: Optional predicate which all returned statements should have • **`obj`**: Optional object which all returned statements should have POST • **`update`**: Optional SPARQL 1.1 update string to be executed. Note that this parameter may be part of the request URL, or in the body of the request with a Content-Type application/x-www-form-urlencoded • **`baseURI`**: Optional base URI to resolve any relative URIs found in uploaded data against. • **`Content-Type`** request header: Where RDF statements are provided for addition to the triplestore. PUT • **`baseURI`**: Optional base URI to resolve any relative URIs found in the uploaded data. • **`Content-Type`** request header: any RDF response type. DELETE • **`subj`**: Optional subject which all returned statements

	should have • **`pred`**: Optional predicate which all returned statements should have • **`obj`**: Optional object which all returned statements should have • **`Content-Type`** request header: any RDF response type.
Formats	No payload. For the POST, PUT, and DELETE methods the X-Transaction response header contains a URL representing this transaction, which is used to refer to this transaction in the logging, provenance and history metadata model.
SAML Authorization	Yes
Description	In the case of a GET request it fetches statements from the repository. For POST, it performs updates on the triplestore, either by adding triples from and RDF document provided in the request, or by executing a SPARQL 1.1 update query provided in the request. With PUT, it updates the triplestore by adding triples from an RDF document provided in the request. DELETE removes the specified triples from the triplestore. As a precaution to prevent the extremely rare case where a user wants to delete all the statements in the repository, or all statements with a particular object, subject or predicate, normal users are required to specify at least two of "**`subj`**", "**`pred`**" or "**`obj`**".

Accessing Previous Triplestore Version Statements

HTTP Method	**`GET`**
HTTP Endpoint	**`/v1/triplestore/versions/`***`{version}`***`/statements`**
Query Parameters	• **`subj`**: Optional subject which all returned statements should have • **`pred`**: Optional predicate which all returned statements should have • **`obj`**: Optional object which all returned statements should have
Formats	None
SAML Authorization	Yes
Description	Fetches statements from the repository in the specified version.

Upload Files to Filestore

HTTP Method	**`POST`**

HTTP Endpoint	**`/v1/filestore`**
Query Parameters	None
Formats	None
SAML Authorization	Yes
Description	Upload a new file to the filestore. Request Headers: • **`Content-Type`**: Mime type of uploaded file. • **`Content-Location`**: Optional absolute or relative URI for source of uploaded file. This is used for informational purposes only during subsequent annotation, and may not be available due to legal constraints upon what data is stored in the warehouse. • **`Last-Modified`**: Optional date and time (HTTP-date) of last modification of file contents. • **`X-Supercedes`**: URL of a file which this file supercedes. Response headers: • **`Location`**: Complete URL for uploaded file. The name of the file is generated internally and is universally unique.

Access Files from Filestore

HTTP Method	**`GET / DELETE`**
HTTP Endpoint	**`/v1/filestore/`***`{filename}`*
Query Parameters	None
Formats	None
SAML Authorization	Yes
Description	Resource representing a file in the filestore. With GET it retrieves the specified file. With DELETE it removes it. Deleting a file should only be done if it contains data which should not be present in the warehouse - for example it has serious errors or it contains personally identifiable information. Normally, if a file is to be replaced with a newer version, the new file should be uploaded through a POST method on "**`/v1/filestore`**", with the **`X-Supercedes`** header set to the URL of the file begin replaced. This is so that analyses which rely upon the previous versions of the file still work. In addition, file deletion also triggers removal of all triples from triplestore produced by that file. This is done globally in the non-reversible manner, so the triples will be removed from all versions of the triplestore.

Access Triples of Files from Filestore

HTTP Method	**GET**
HTTP Endpoint	**/v1/filestore/***{filename}***/triples**
Query Parameters	None
Formats	N-Triples
SAML Authorization	Yes
Description	Resource representing the raw triples extracted from the file upon submission. It retrieves the triples of the specified file, in the N-triples format

Access Ontologies of Files from Filestore

HTTP Method	**GET**
HTTP Endpoint	**/v1/filestore/***{filename}***/ontology**
Query Parameters	None
Formats	N-Triples
SAML Authorization	Yes
Description	Returns the ontology file (in OWL format) representing the triples extracted from the file upon submission, and used in the "/v1/filestore/{filename}/triples".

Access Annotation Description Files for Files from Filestore

HTTP Method	**GET/PUT**
HTTP Endpoint	**/v1/filestore/***{filename}***/annotationDescription**
Query Parameters	None
Formats	None
SAML Authorization	Yes
Description	Returns the ontology annotation file of the given fileName. If the PUT method is used a new annotation file is uploaded. This triggers a call to the Data Translator library to transform raw triples into the data to be stored in the triplestore. If the annotation description subsequently changes, the previously added triples are removed prior to addition of the new triples

Access Image Files from Imagestore

HTTP Method	**GET**

HTTP Endpoint	**`/v1/imagestore/`***`{SOPInstanceUID}`*
Query Parameters	None
Formats	None
SAML Authorization	Yes
Description	Retrieves the specified image file

Access Triples for Image Files from Imagestore

HTTP Method	**`GET`**
HTTP Endpoint	**`/v1/imagestore/`***`{SOPInstanceUID}`***`/triples`**
Query Parameters	None
Formats	N-Triples
SAML Authorization	Yes
Description	Resource representing the raw triples extracted from the image upon submission. It retrieves the triples of the specified image file, in the N-triples format.

Non functional requirements

The Data Warehouse should have access to a large storage. However, it cannot be said in advance how much space will be needed because it is always depending on the amount of data that is going to be put in there but, usually, the more available space, the better. In addition, as explained in all operations, Data Warehouse is embedded in the security framework and it needs authorised requests at all times by passing the SAML token in the "Authorization" HTTP header.

Installation instructions

Required software:

- Operating System: Linux (Ubuntu 12.04)
- Persistence Layer: OpenRDF Sesame 2.x and Redis 2.x
- Application Container: Apache Tomcat 6 or 7
- Java 7 Runtime Environment

Installation Steps:

- Install and start Tomcat (regular installation, nothing special)
- Install and start Redis (regular installation, nothing special)
- Deploy OpenRDF Sesame (the workbench too) in Tomcat
- Create a native repository in OpenRDF with name "**`dwh`**" using the OpenRDF Workbench
- Create the following folders in "**`/var/lib/`**":

```
    - dwh/
    - dwh/files/
    - dwh/files/dwh/
    - dwh/audit-logs/
```

- Put "**DataTranslationAPIConfig.xml**" file in your preferred location.
- Create the configurations file following the template below:

```
openrdf-sesame =
repository-id = "dwh"
authz-service =
disable-audit-log = false
dwh.repository-id = "dwh"
dwh.authz-service =
dwh.audit-log-path = "/var/lib/dwh/audit-logs"
dwh.backend-options =  {:root "/var/lib/dwh"}
conf-dt =
```

- Deploy Data Warehouse WAR "**dwh.war**" file in Tomcat

4.2 TTP Services

Upload file / Create processing Request

HTTP Method	**POST**
HTTP Endpoint	**/services/rest/processingRequest/?name=***{fname}*
Query Parameters	None
Formats	Text
SAML Authorization	Yes
Description	Uploads a new (pseudonymized) file into the TTP using the specified name It returns the processing request id. TTP should have knowledge of the schema of the uploaded file, otherwise processing will fail.

Get Request status

HTTP Method	**GET**
HTTP Endpoint	**/services/rest/processingRequest/status/***{id}*
Query Parameters	None
Formats	Text/XML
SAML Authorization	Yes
Description	Accesses the processing status of the uploaded file..

Example Output:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<UploadStatus state="DELIVERED" name="data.csv" id="1530155"/>
```

Get All Requests

HTTP Method	**`GET`**
HTTP Endpoint	**`/services/rest/processingRequest/status`**
Query Parameters	None
Formats	Text/XML
SAML Authorization	Yes
Description	Accesses the processing status of all the uploaded files. File names are not unique so multiple response can thus be returned

Example Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<UploadStatuss>
<UploadStatus state="DELIVERED" name="data.csv" id="1800180"/>
<UploadStatus state="DELIVERED" name="data.csv" id="1670167"/>
<UploadStatus state="DELIVERED" name="data.csv" id="1530155"/>
<UploadStatus state="CONFIRMED_ERROR" id="1530154"/>
<UploadStatus state="CONFIRMED_ERROR" id="1530153"/>
<UploadStatus state="CONFIRMED_ERROR" id="1450145"/>
<UploadStatus state="CONFIRMED_ERROR" id="1370137"/>
</UploadStatuss>
```

4.3 Ontology Annotator

The Ontology Annotator is a **web-based** tool that can be accessed through the p-medicine portal. It was designed to allow end users to specify annotations of databases for their semantic integration in the p-medicine platform.

The Ontology Annotator is an annotation tool for specifying semantic relationships between elements of the schema of a database with elements of an ontology (in the case of p-medicine, the HDOT ontology). Users can create "annotation projects" for their database, in which they can browse their database schema and the HDOT ontology and specify the semantic relations between them that will enable the automatic translation of the database into an HDOT-compliant form. The annotation of a database can be done in different sessions (users can save their work and recover it in subsequent accessions), can be shared with other users (several users can collaborate in one annotation project), and, when finished, can be submitted to the p-medicine platform. Once the annotation of a database is submitted, the corresponding database will be automatically integrated into the p-medicine Data Warehouse.

The Ontology Annotator is supported by the HDOT ontology. This ontology is graphically represented to users during the annotation projects. The Ontology Annotator also makes use of the TTP services to ensure that all operations are secure. Finally, the Ontology Annotator communicates with the Data Warehouse (through the TTP services) to upload the annotations defined by the users.

Ontology annotator provides some search operations and management of the projects and users like:

- Login.
- Create project.
- Open project.
- Delete project.
- Add new user to project.
- Create database view.
- Create HDOT view.
- Create annotation entry.
- Submit annotation.

Non functional requirements

The Ontology Annotator performs user authentication, by accessing the TTP services.

Installation instructions

Ontology Annotator is a web-based tool which must be deployed in an application contained with JSP capabilities (e.g. Tomcat version 7). The tool also requires JDK version 7.

The Ontology Annotator also employs an SQL database as persistence layer. This database must be accessible by the tool (either locally or remotely).

Detailed installation instructions can be found in the p-medicine wiki.

4.4 Data Translator

The Data Translator is a Java-based API capable of transforming data into an HDOT-compliant form. It provides a unique service, for translating data from its original format to the format of HDOT. For performing this operation, the Data Translator receives the original data in form of N-Triples (http://www.w3.org/TR/n-triples/) and one annotation description, and as a result produces another set of N-Triples, based on the HDOT structure. The annotation description received is actually the identifier of the annotation file that must guide the translation process. This annotation file is stored in the Ontology Annotator database.

The Data Translator is used solely by the p-medicine Data Warehouse (DW). The DW invokes the Data Translator whenever a new database is uploaded or updated, or when the annotation for an existing database is updated. It also collaborates with the Ontology Annotator, for retrieving the required annotation files.

Interface

The Data Translator provides a Java Library (JAR) interface for accessing its translation service. The Data Translator includes two different operations: **configure** and **translateData**

Operation: configure

This operation configures the Data Translator with the necessary parameters. The configure method is contained in the **`es.upm.gib.datatranslator.DataTranslationService`** class, and is accessible as a public static method. The configure operation has three (overloaded) methods in this class (the client can invoke the one that finds more convenient, they all produce the same result). Their signature is the following:

```
public static void configure(String configurationFileName)
throws IOException;
```

```
public static void configure(File configurationFile) throws IOExcept
ion;
public static void configure(URL configurationFileURL) throws IOExce
ption;
```

which only differ in the parameter received.

The input to the configure operation is an XML file containing information for properly initializing the Data Translator. Each of the three overloaded methods receives this XML file as a different type:

- **`String configurationFileName`**: this parameter provides a local path to the XML configuration file
- **`File configurationFile`**: this parameter provides a **`java.io.File`** object pointing to the XML configuration file
- **`URL configurationFileURL`**: this parameter provides an URL which gives access to the XML configuration file

Below is an example of a valid configuration file:

```
<?xml version="1.0"?>
<DT_Service>
    <!-- Location of the HDOT files for initial loading. Options: -->
    <!-- i) <HDOTLocation type="directory">path</HDOTLocation> -> loads the
HDOT files from a local directory, specified by path -->
    <!-- ii) <HDOTLocation type="URL">URL</HDOTLocation> -> loads the HDOT
files from a specific URL, specified by URL -->
    <!-- iii) <HDOTLocation type="source"/> -> loads the HDOT files from
the HDOT project page -->
    <HDOTLocation type="source"/>

    <!-- Names of the actual HDOT files to load -->
    <!-- HDOT is divided in several files, which correspond to its
different modules. This element indicates the names of the actual files to
load -->
    <!-- These names are contatenated to the HDOT location, provided in the
previous element -->
    <!-- To see a full list of files composing HDOT, visit its project page
at http://code.google.com/p/hdot/source/browse/trunk/ -->
    <HDOTFiles>
        <HDOTFile>swo_inferred_pMed.owl</HDOTFile>
        <HDOTFile>hdot_core.owl</HDOTFile>
        <HDOTFile>hdot_KLt.owl</HDOTFile>
        <HDOTFile>doid_import.owl</HDOTFile>
        <HDOTFile>hdot_pfm.owl</HDOTFile>
        <HDOTFile>hdot_pem.owl</HDOTFile>
        <HDOTFile>hdot_OXt.owl</HDOTFile>
        <HDOTFile>hdot_bsds.owl</HDOTFile>
        <HDOTFile>hdot_dicom.owl</HDOTFile>
        <HDOTFile>swo_inferred_pMed.owl</HDOTFile>
        <HDOTFile>swo_inferred_v0.5.owl</HDOTFile>
        <HDOTFile>hdot_pm.owl</HDOTFile>
    </HDOTFiles>

    <!-- URL where the DW is deployed, including its version (v1) -->
    <DWURL>https://schroedinger.chem.ucl.ac.uk:8443/dwh/v1/</DWURL>

    <!-- location of the keystore file containing the DW certificate
(needed to properly access the DW) -->
```

```
    <!-- can be placed in a local directory (type="directory") or in an URL
(type="URL"). The password attribute indicates the password for accessing
the keystore. -->
    <!-- The text value of the tag indicates either the local directory or
the URL -->
    <keystore type="directory" password="123456">c:\myKeys.kst</keystore>
</DT_Service>
```

Operation: translateData

This operation performs the translation of the provided data into an HDOT-compliant form. The **`translateData`** method is contained in the **`es.upm.gib.datatranslator.DataTranslationService`** class, and is accessible as a public static method. Its signature is the following:

```
public static Iterator translateData(InputStream rawTriples,
InputStream annotationDescription, String baseURI) throws Exception;
```

The translateData operation receives three parameters:

- **`InputStream rawTriples`**: an InputStream which must be open and provide access to the N-TRIPLES content of the file to translate
- **`InputStream annotationDescription`**: an InputStream which must be open and provide access to the annotation identifier to use for this translation
- **`String baseURI`**: the base URI of the file to translate (required to properly extract the data from the file)

The **`translateData`** operation returns an Iterator over **`es.upm.gib.owlbasicmodel2.NTripleGenerator.Triple`** objects. Each of these objects is a simple collection of three Strings: origin, property and destination (each with its corresponding getter), which form an RDF triple. The returned iterator allows recovering all the HDOT-compliant triples generated during the translation process.

Installation instructions

The Data Translator must be imported as a Java library in the project that requires using it. It requires **Java 1.7** or higher to run. Some specific details:

- It is OS-independent
- There is no persistence layer required. No data is stored
- There are no specific HPC requirements. The Data Translator can run in any average machine
- **Importing the Data Translator into a project**
- **Maven configuration**

The Data Translator can be imported into a maven-enabled project, by specifying the following dependency:

```
es.upm.gib.datatranslator
datatranslator
1.0
```

4.5 ALGA-C Profiling Service

This service is part of the Patient Empowerment suite of services and provides access to the results of the processing of a patient's answers after he/she has completed the ALGA-C questionnaire.

The service publishes a single endpoint URL for retrieving the questionnaire results of a single patient:

Operation: Get Patient Profile

HTTP Method	`GET`
HTTP Endpoint	`/getPatientResults`
Query Parameters	`patientId` : the patient identifier, uniquely identifying the patient in the Patient Empowerment context
Formats	JSON[5]

This endpoint returns the results of the questionnaire for each of the following categories:

- Perceived Health State
- Physical_Health State
- PsychoSocial Aspects
- Cognitive Aspects
- Psychological Aspects

And the following subcategories (per category):

Category	Subcategory
Cognitive Aspects	Cognitive Closure
	Memory Attention
	Rumination
Perceived Health State	Perceived Health State
Physical Health State	Physical Health State
PsychoSocial Aspects	Body Image
	Sexual Problems
	Self Efficacy

In the case that the same patient has filled in the questionnaire multiple times in the past, the service returns the most recent results.

Example Output:

[5] JavaScript Object Notation, http://json.org/

The returned message is in JSON format. It is actually an array (sequence) containing objects of the following form:

```
{
    "score": 1,
    "z_score": -1.414730358868464,
    "recommendationText":   "<br>Explanation:   High   values   are
generally associated with a negative mood state and to problems with
intimate   relationship.   <br>Recommendation:   highlight   eventual
organic-therapeutic  consequences  and  distinguish  those  from  the
psychological ones.",
    "numberOfQuestions": 2,
    "pointsSum": 2,
    "subcategory": "Sexual_Problems",
    "category": "PsychoSocial_Aspects"
 }
```

The **numberOfQuestions** field provides the number of questions that this category/subcategory pair contains while the **z_score** contains the (normalised to zero mean) score of the patient in this set of questions. This score can be interpreted as follows:

- Values between -1 and 1 are considered "safe" or "normal"
- Values between -2 and -1 or 1 and 2 are "cautious" i.e. somewhat divergent from the normal case
- Values below -2 or above 2 are on the critical level

The **recommendationText** contains an explanation for the non-normal setting and some recommendations. These are always the same irrespective of the actual **z_score**.

4.6 Drug – Drug Interaction Service

This is a RESTful web service for retrieving drug related information. It offers the following functionality:

- Get information for a specific drug
- Return potential interactions among two given drugs.

Operation*:* Get Drug information

HTTP Method	**GET**
HTTP Endpoint	**/getDrugInformation**
Query Parameters	**drugName** : the name or synonym of a drug
Formats	JSON

This endpoint returns information about the given drug, such as the following:

- Name
- ID
- Synonyms
- Categories
- Dosages
- Brand names
- Drug interactions

- Food interactions

Example Output:

```
{
  "drugbank_id": "DB00471",
  "name": "Montelukast",
  "brands": [
    "Montair",
    "Singulair",
    "Singular"
  ],
  "synonyms": [],
  "description": "Montelukast is a leukotriene receptor antagonist
(LTRA) used for the maintenance treatment of asthma and to relieve
symptoms of seasonal allergies. It is usually administered orally.
Montelukast blocks the action of leukotriene D4 on the cysteinyl
leukotriene receptor CysLT1 in the lungs and bronchial tubes by
binding to it. This reduces the bronchoconstriction otherwise caused
by the leukotriene, and results in less inflammation. Because of its
method of operation, it is not useful for the treatment of acute
asthma attacks. Again because of its very specific locus of
operation, it does not interact with other allergy medications such
as theophylline. Montelukast is marketed in United States and many
other countries by Merck & Co. with the brand name Singulair®. It is
available as oral tablets, chewable tablets, and oral granules. In
India and other countries, it is also marketed under the brand name
Montair®, produced by Indian company Cipla.",
  "categories": [
    "Anti-Asthmatic Agents",
    "Antiarrhythmic Agents",
    "Leukotriene Antagonists"
  ],
  "dosages": [
    {
      "form": "Granule",
      "route": "Oral",
      "strength": "No strength information"
    },
    {
      "form": "Tablet",
      "route": "Oral",
      "strength": "No strength information"
    }
  ],
  "drug_interactions": [
    {
      "drug": "DB01124",
      "name": "Tolbutamide",
      "description": "Tolbutamide, a strong CYP2C9 inhibitor, may
decrease the metabolism and clearance of Montelukast. Consider
alternate therapy or monitor for changes in Montelukast therapeutic
and adverse effects if Tolbutamide is initiated, discontinued or
dose changed. "
    }
  ],
  "food_interactions": [
    "Take without regard to meals."
  ]
}
```

The drug should exist. In case of an unknown drug the service returns an error:

```
{
"Error":"Drug with the specified name is not found in the database!"
}
```

Operation: Get Drug-Drug Interaction

HTTP Method	`GET`
HTTP Endpoint	`/getDrugToDrugInteractions`
Query Parameters	**`drugNameA`** : the name or synonym of the first drug **`drugNameB`** : the name or synonym of the second drug
Formats	JSON

It returns the description of a potential interaction between two drugs

Example Output:

```
The prostacyclin analogue, Treprostinil, increases the risk of
bleeding when combined with the anticoagulant, Lepirudin. Monitor
for increased bleeding during concomitant thearpy.
```

In the case that no interaction found, the service returns:

```
No Interactions are found between the two drugs!
```

4.7 "Donor's Tool" Consent Access Service

This is a RESTful web service for retrieving information about the available "donation projects" and their consent documents and related material (see paragraph 3.4.4). It offers the following functionality:

- Get all projects
- Get general information for a specific project
- Get all the relevant documents for a specific project.

Operation: Get All Projects Ids

HTTP Method	`GET`
HTTP Endpoint	`/DonorsTool/getAllProjectIdsAndNames`
Query Parameters	

Formats	JSON

It returns a list of the available project ids and their names.

Example Output:

```
[
 {"projectId": 1, "name": "Test project 1"},
 {"projectId": 2, "name": "Test project 2"}
 ...
]
```

Operation*:* **Get General Project Information**

HTTP Method	**`GET`**
HTTP Endpoint	**`/DonorsTool/getGenericProjectInformation`**
Query Parameters	**`projectId`** : the id of a specific project
Formats	JSON

It returns the details of the specific project such as the its name, summary, and contact person information.

Example Output:

```
{
 "projectId": 6, "name": "i", "summary":"i", "lang": "en",
 "contactPerson": {
    "name": "John Doe",
    "email": "doej@example.com",
    "phone": "555-555-5555"
 }
}
```

Operation*:* **Get Documents of a Project**

HTTP Method	**`GET`**
HTTP Endpoint	**`/DonorsTool/getConcentFilesInformationByProjectId`**
Query Parameters	**`projectId`** : the id of a specific project
Formats	JSON

It returns (links to) all relevant documents for the given project.

Example Output:

```
[
 {
  "projectId": 6,
  "consentDescription": "...",
  "doc": "http://example.com/6/1.pdf"
  }
  ...
}
```

4.8 Workbench

The Workbench is a tool registry and repository. The basic functionality it offers is the discovery of tools and services based on some given criteria.

Tool searching

HTTP Method	`GET`
HTTP Endpoint	`/v1`
Query Parameters	• `qcat`: a “strong” filter for the categories the returned tools should belong • `q`: a “weak” filter for searching the names and descriptions of the tools using “full-text” search capabilities • `qtag`: a “weak” filter for (full-text based) searching the user supplied tags. Can be used multiple times.
Formats	JSON
SAML Authorization	Not required
Description	Searching for tools. The client can use a number of key - value pairs in the query part of the URI in order to set search filters. The filters can be "strong" or "weak". Strong filters must be satisfied for a tool to be returned in the results. "Weak" filters are not filters actually, they are used for sorting the results so that tools that satisfy them are returned first.

Example output:

For a request searching for the word “miRNA” in the names and descriptions of tools (…**`/v1?q=mirna`**) we can get the following output:

```
{
  "count": 13,
  "total": 13,
  "limit": 1000,
  "start": 0,
  "items": [
    {
      "img": null,
      "score": 1,
      "name": "MiRDeep",
      "ratings": null,
      "@type": "tool",
      "@id": "v1/012dbea9-d4dc-42c0-bf82-49e8813242fe",
      "user_likes": null,
      "description":  "Discovering  known  and  novel  miRNAs  from  deep
sequencing data",
      "tag": ["mirna"],
      "category": ["Sequence Analysis"],
      "tags_per_role": [
        {
          "role": "ADMINISTRATOR",
          "tags": ["mirna"]
        }
      ]
    },
    {
      "img": null,
      "score": 1,
      "name": "SeqBuster",
      "ratings": null,
      "@type": "tool",
      "@id": "v1/0365a3cd-e0b2-4387-a7e8-5780e035a416",
      "user_likes": null,
      "description": "SeqBuster, a web-based bioinformatic tool offering a
custom analysis of deep sequencing data at different levels, with special
emphasis on the analysis of miRNA variants or isomiRs and the discovering
of new small RNAs. ",
      "tag": ["Small RNA transcriptome","miRNA "],
      "category": ["Sequence Analysis"] ,
      "tags_per_role": [
        {
          "role": "ADMINISTRATOR",
          "tags": ["Small RNA transcriptome","miRNA"]
        }
      ]
    },
  //… rest omitted for brevity
  ]
}
```

Tool retrieval

HTTP Method	**GET**
HTTP Endpoint	**/v1/***{toolId}*
Query Parameters	None
Formats	JSON
SAML Authorization	Not required
Description	Get the metadata for a specific “tool”.

Example Output:

```
{
  "@type": "tool",
  "@id": "v1/fd0cfbc5-74f0-4389-b5b3-01fde1c60c40",
  "id": "fd0cfbc5-74f0-4389-b5b3-01fde1c60c40",
  "file": null,
  "publisher": null,
  "description": "Basic graphic utilities for visualization of genomic
data. The biovizBase package is designed to provide a set of utilities,
color schemes and conventions for genomic data. It serves as the base for
various high-level packages for biological data visualization. This saves
development effort and encourages consistency.",
  "tag": ["Bioinformatics", "Infrastructure",
        "Preprocessing", "Software", "Visualization"],
  "category": ["Microarrays"]
  "license": null,
  "creator": null,
  "img": null,
  "source_code": null,
  "semterm": [
    "http://edamontology.org/data", "http://edamontology.org/data_0006",
    "http://edamontology.org/topic", "http://edamontology.org/topic_0209",
    "http://edamontology.org/topic_3176"],
  "name": "biovizBase",
  "technology_type": null,
  "webpage":
"http://www.bioconductor.org/packages/release/bioc/html/biovizBase.html",
  "operation": [
    {
      "@id": "v1/op/546c193c-ff54-4c82-98ee-65a50ad6fbda",
      "@type": "operation",
      "name": "biovizBase",
      "description": null,
      "semterm": []
      "input": [
        {
          "name": "GenomicFeatures",
          "description": null,
          "required": true,
          "data_type": null,
          "@id": "v1/in/00846052-8db6-40c8-9316-93112620abe3",
          "@type": "input",
          "semterm": null
        },
        {
          "name": "grDevices",
          "description": null,
          "required": true,
          "data_type": null,
          "@id": "v1/in/08e97b08-6636-4577-890e-840e50013fc8",
          "@type": "input",
          "semterm": null
        },
        //… rest input parameters omitted for brevity
      ],
      "output": [
        {
          "name": "vizualization",
          "description": null,
          "data_type": null,
          "@id": "v1/out/f87fe09d-e966-4fb0-8b79-a71355a69b5e",
          "@type": "output",
          "semterm": null
        }
      ]
    }
  ],
  "tags_per_role": [],
  "os": null,
  "ratings": null,
  "user_likes": null
}
```

Tool Update

HTTP Method	`PUT`
HTTP Endpoint	`/v1/{toolId}`
Query Parameters	None
Formats	JSON (in request)
SAML Authorization	Yes
Description	Update the metadata for a specific “tool”. The client should submit the complete description of the tool, i.e. its operations, inputs, outputs, etc. and the existing definition in the workbench will be completely replaced.

Update the “rating” of a tool

HTTP Method	`POST`
HTTP Endpoint	`/v1/{toolId}/ratings`
Query Parameters	None
Formats	-
SAML Authorization	Yes
Description	Update the ratings for a specific “tool”. In the tool repository it is registered that the current principal user (extracted from the SAML token) has “liked” the given tool. Therefore the number of “likes” for this tool is increased by one.

Update the role specific tags of a tool

HTTP Method	`POST`

HTTP Endpoint	**/v1/***{toolId}***/tags/***{role}*
Query Parameters	None
Formats	JSON (in request)
SAML Authorization	Yes
Description	Update the tags of he given “tool” for the specific role. The request contains the tags to be added in JSON formatted array of strings.

Non functional requirements

User roles will be used to regulate actions performed within the workbench. Three user roles are used to regulate actions within the workbench: CLINICIAN, SCIENTIST, and ADMINISTRATOR. These roles match the existing user roles of the P-Medicine's portal:

Portal Role	Workbench Role	User groups
CLINICIAN	**CLINICIAN**	Clinicians, physicians
CLINICAL TRIAL LEADER	**CLINICIAN**	Clinical trial chairman, clinical trial designers
SCIENTIST	**SCIENTIST**	Researchers from the scientific community (mathematicians, physicists, bioinformaticians, computer scientists, etc.)
PMED ADMINISTRATOR	**ADMINISTRATOR**	Administrator of the p-medicine portal (responsible for user management, user permissions, community management, etc.)
COMMUNITY ADMINISTRATOR	**ADMINISTRATOR**	Administrators of the communities created in the p-medicine portal
DEVELOPER (IT)	**ADMINISTRATOR**	Developers of tools and services for p-medicine

Actions within workbench are regulated by user roles. These actions are listed below (the roles in the table below, correspond to Workbench's User Roles):

User Role	Actions
No role required	Can search for tools
At least one role	Can register a tool
At least one role	Can add new or edit existing tool tag.
ADMINISTRATOR	Can confirm that some registered tool is valid.

Furthermore the system utilizes these roles in order to assist user during the tool discovery phase. Thus, the provided custom tags are displayed in certain priority that depends on user's roles. i.e. tags that have been created by clinicians will be displayed first in case that the current user has the role of clinician. In case that the user does not have any role, will still be able to see the tags, but in mixed sequence. In case that the user has multiple roles, the following priority is taken into account:

- CLINICIAN
- SCIENTIST
- ADMINISTRATOR

This priority is used for determining which tags to be displayed first and which role to be used as reference when a user creates a new or edit's an existing tag (users with no roles cannot delete, edit or create new tags). The Workbench Portlet in the Portal can also be accessed by non-authenticated users, but in this case the functionality is limited. For example only authenticated users can make one tool as their “favourite”.

Therefore, the server makes use of the p-medicine security framework to achieve two goals:

- Restrict access to the operations that perform server side, i.e. any POST, PUT, DELETE request. This means that retrieving the workbench contents (e.g. through search) is always allowed and unauthenticated requests (i.e. GETs without SAML token in the Authorization HTTP header) are always allowed. Of course the requests coming from the workbench portlet will always carry user authentication information.
- Support "personalization". When an authenticated user is the requester of the information returned by its REST API, the server can take advantage of the role information associated with this user (as retrieved by the SAML token). For example for each tool we can have tags associated with the role that the user who submitted them had. This way we can return role specific tags (e.g. clinician related tags versus bioinformaticians ones)

Installation instructions

Software requirements:

- The tools are stored in a PostgreSQL database so you need to install it first. the recommended versions are in the 9.X series, 9.3.3 being the latest at the time of this writing.
- Java Runtime Environment 6 or newer.

The server is provided as a single JAR file and no other application server (e.g. Tomcat) is needed. In order to test it please do as follows:

- Bootstraping the database.
- Change the configuration file to specify TCP port to use, etc.
- Launch it by issuing in the commend line: **`java -jar wbench-0.5-standalone.jar`**
- Use the browser to visit **`http://localhost:<port>/v1?q=analysis`**

5 Non functional requirements

The most fundamental quality of the p-medicine architecture is security. The data upload and management is very complex due to this emphasis on the preservation of patient's privacy and anonymity.

Next to that is the issue of usability and user friendliness. This is an ongoing task and there were a number of usability workshops, the most recent of the 7th progress meeting in April 2014.

Finally after the experts' recommendations in the 2nd review of the project we have made some extensive analysis of the system's performance focusing on specific components of the architecture. The methodology and the results can be found in Appendix A.

6 Deployment

The p-medicine platform will be distributed along many computational nodes due to its complexity, functionality, and heterogeneity of components. A test deployment comprising most of the platform's components is available in a private *cloud* environment hosted by partner PSNC in Poland since they have access to the highly efficient and available storage services provided by National Data Storage (NDS), which is part of the EUDAT (European Data Infrastructure, http://www.eudat.eu/).

The cloud offers the flexibility to adapt to increased demands for computation and storage and therefore is the ideal infrastructure for deploying a complex system such as p-medicine. The primary emphasis of the p-medicine architecture is the handling of large data sets and therefore storage is the primary concern. The Cloud storage in p-medicine environment is needed primarily by the Data Warehouse for the management of files and RDF data. The other scenarios are related to data mining workflows that can use cloud storage for intermediate computation results, and oncosimulator application executed in a dedicated cluster environment (not shown here).

The current prototype of the p-medicine platform deployed in Poland is shown below:

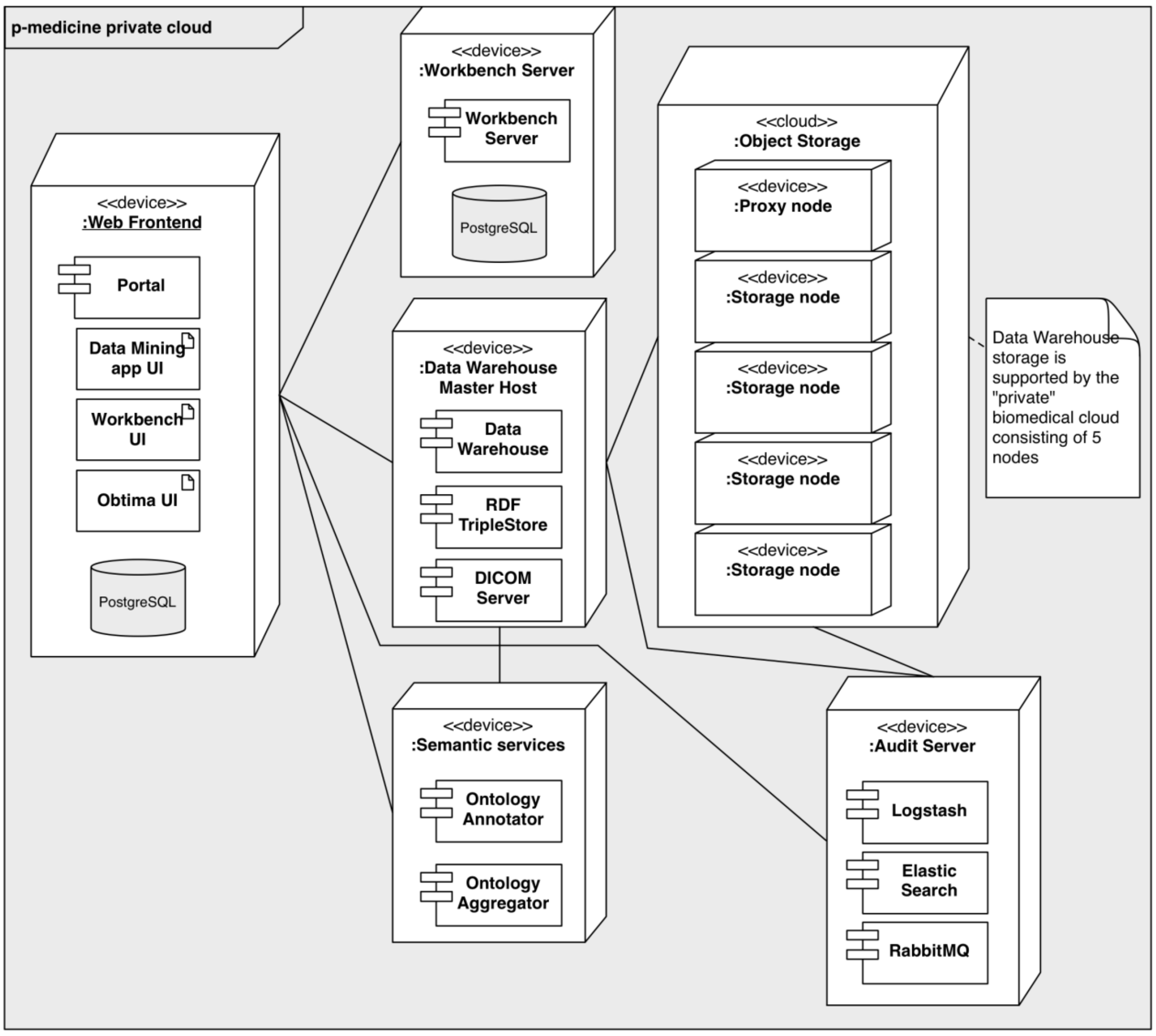


Figure 26 The deployment for the prototype system

The only services currently not installed in the p-medicine cloud are the security related ones (e.g. CATS, the pseudonymisation service) that are hosted by partner Custodix on their premises.

The configuration of the virtual machines and relevant p-medicine as well as third party software is shown in the next Table.

Node(s) Functionality	Components	Configuration
Portal	**Liferay**[6] **portal** within Apache Tomcat[7] and apache, PostgreSQL[8], Taverna. The portal hosts various tools for accessing the p-medicine services (e.g. Workbench, Data Mining etc) as *portlets*.	2 CPU, 10 GB RAM, 110 GB storage
Data warehouse	**Data Warehouse**, Redis[9], dcm4chee DICOM server[10], OWLIM triplestore[11]	3 CPU, 10 GB RAM, 4 TB storage
Semantic Services	**Ontology Annotator** and **Ontology Aggregator** with Apache Tomcat	1 CPU, 2 GB RAM, 10 GB storage
Workbench	**Workbench** Server, nginx web server[12], PostgreSQL	1 CPU, 2 GB RAM, 30 GB storage
OpenStack Object Storage	OpenStack's Object Storage (Swift[13])	• 1 Proxy node: 4 CPU, 10 GB RAM, 10 GB storage • 4 storage nodes: 4 CPU, 10 GB RAM, 16 TB storage
Auditing Server	RabbitMQ[14], ElasticSearch,[15] logstash[16] and Apache Tomcat with kibana[17]	1 CPU, 2 GB RAM, 110 GB storage

All machines run Ubuntu Linux 14.04 LTS[18].

[6] http://www.liferay.com/
[7] http://tomcat.apache.org/
[8] http://www.postgresql.org/
[9] http://redis.io/
[10] http://www.dcm4che.org/
[11] http://www.ontotext.com/owlim/
[12] http://nginx.org/
[13] http://swift.openstack.org/
[14] http://www.rabbitmq.com/
[15] http://www.elasticsearch.org/
[16] http://logstash.net/
[17] http://www.elasticsearch.org/overview/kibana/

The current prototype fits more than enough the internal needs of the project but but this is an initial version of the platform and a production level platform will probably need more resources. The use of the cloud allows such “scale out”. From the beginning, all resources are monitored and in case of higher resource usage additional nodes will be added. Of note, the use of REST architectural style and its stateless principle affords increase in scalability of the p-medicine services through load balancing and partitioning with relatively ease. The reader can refer to Appendix A for more details about the performance and scalability characteristics of prominent p-medicine tools.

[18] http://www.ubuntu.com/

7 Architecture Compliance

The p-medicine platform intends to be open so that new components can be attached to its architecture. Of course such openness should not be unconditional, i.e. there should be certain requirements met by an external component in order to be compatible with the p-medicine platform. Furthermore, there can be different levels of compatibility between the designed system and a third party component. Of note, The Open Group Architecture Framework (TOGAF[19]) introduces different terms (e.g. “conformant”, “compliant” etc.) and provides detailed definitions of those terms so as to describe the possible degrees of compatibility between the architecture and an implementation of this architecture (Figure 27).

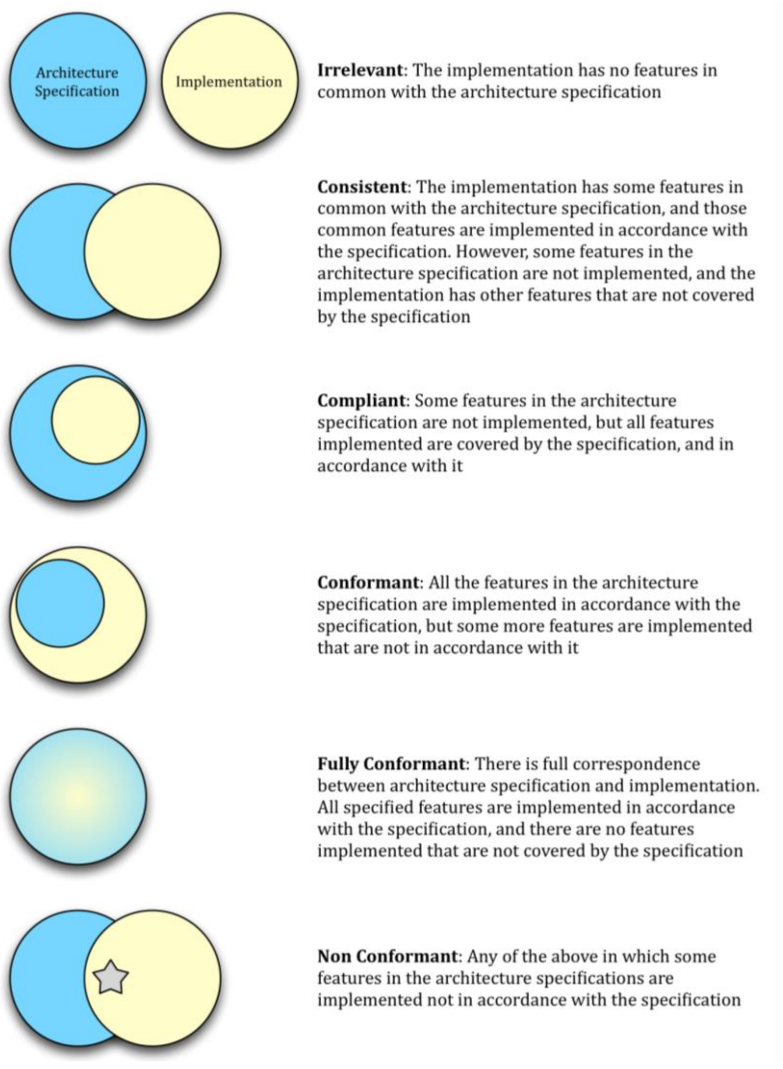

Architecture Specification
Implementation
Irrelevant: The implementation has no features in common with the architecture specification
Consistent: The implementation has some features in common with the architecture specification, and those common features are implemented in accordance with the specification. However, some features in the architecture specification are not implemented, and the implementation has other features that are not covered by the specification
Compliant: Some features in the architecture specification are not implemented, but all features implemented are covered by the specification, and in accordance with it
Conformant: All the features in the architecture specification are implemented in accordance with the specification, but some more features are implemented that are not in accordance with it
Fully Conformant: There is full correspondence between architecture specification and implementation. All specified features are implemented in accordance with the specification, and there are no features implemented that are not covered by the specification
Non Conformant: Any of the above in which some features in the architecture specifications are implemented not in accordance with the specification

Figure 27 The levels of the architecture conformance according to TOGAF

In our case the p-medicine platform is fully conformant to its architecture “by definition” (since it was developed in accordance with it) but the important question is: What are the requirements for a new tool, which could be developed outside of the p-medicine, to be compliant with the p-medicine platform, i.e. to implement some of its features while still conforming to its architectural constraints?

[19] TOGAF®, an Open Group standard: http://www.opengroup.org/subjectareas/enterprise/togaf/

Deliverable 8.6.1 ("Integration Guidelines and Monitoring of tools and services") describes three "integration maturity" levels aptly named Bronze, Silver, and Gold. These levels, shown in the next table, refer to the provision of programmatic interfaces for the accessing the component in question, and Deliverable 8.6.1 recommends the adoption of at least the Silver level. This means that potential third party tools should at least provide a well-described programmatic interface over secure transport for the exchange of data in the data formats adopted by the consortium.

Architectural View	Legacy	Bronze	Silver	Gold
Programming and Messaging Interfaces.	- No programmatic interfaces to the system are available. Only local data files in a custom format can be read. - Data transfer mechanisms are implemented only on an ad hoc basis.	- Programmatic access to data from an external resource is possible.	- Well-described APIs, approved by the p-medicine Architecture Board, provide access to data. - Electronic data formats corresponding to a registered domain model approved by the p-medicine Architecture Board are supported wherever messaging is indicated by the use cases. - Messaging protocols approved by the p-medicine Architecture Board are supported wherever messaging is indicated by the use cases. - Secure services must use the p-medicine security guidelines mechanisms for authentication, trust management, and communication channel protection.	- All features of silver, plus: - APIs are exposed as operations of a web service; Object-Oriented client APIs are available for invoking those operations. -Service operations use XML as data exchange format, and are invoked using standardized protocols and communication channels. - Services provide public access to p-medicine standardized service metadata and have capability to register it with the p-medicine Tool/Service Repository.

Of course these are generic guidelines that can be adapted in accordance to the specific application area of personalized medicine. As described in Sections 2.2 and 2.4, the p-medicine platform covers domains such as clinical decision support, computational cancer modelling, data mining, etc. that have different requirements or even operate in different contexts, e.g. clinical treatment domain versus clinical research domain. Despite these differences there are two main points in the compatibility checklist for a candidate new p-medicine component or application:

- Conformance to the security infrastructure and the related guidelines. This means that the component under consideration should use the authentication, authorization, confidentiality, and trust mechanisms of the p-medicine platform (Section 3.1). Concretely, the component should use SAML tokens, preferably over REST/HTTPS message exchange channels, and contacting the designated p-medicine Identity Provider (IdP) server. In the case where the new component provides a data upload functionality (for example, an external data source "feeding" the p-medicine data warehouse), it should additionally comply to the secure data management requirements: use the pseudonymization infrastructure and more specifically the

CATS and TTP services (Sections 3.2.1 and 4.2) in order to ensure the privacy of the patient data.

- Conformance to the semantics and the ontology based annotation of the data managed in p-medicine. A data provider should use the Ontology Annotator (Section 3.2.2) to provide proper semantic annotations to the uploaded data and then register those annotations into the data warehouse so the correct transformations are performed and the data are integrated appropriately. On the other hand, an application, which intends to reuse data managed by p-medicine, should use the semantically rich, SPARQL based interface of the Data Warehouse in order to retrieve the data.

As already mentioned these are relatively universal requirements for connecting the p-medicine platform. On specific occasions more restrictions apply. For example, the use of a new computational model for in-vivo tumor evolution requires the use of certain software libraries and development techniques in order to be used in the p-medicine computational modelling infrastructure and take advantage of high performance (HPC) or cloud computing. The detailed integration requirements and compatibility checking instructions on these domain specific cases are outside the scope of this document.

8 Conclusions

The p-medicine architecture is defined a complex system of loosely interconnected subsystems. The different subsystems correspond to application areas of the personalized medicine vision, such as clinical decision support, patient empowerment, computational modelling etc. The architectural components comprising the p-medicine platform interact using well-defined programmatic interfaces based on HTTP web services. Although the platform may appear loose and unrestricted its components are bind together using the strict security guidelines and the semantic infrastructure as the primary integration mechanisms. A prototype version of the system has been deployed in a private cloud environment in Poland.

9 References

[1] N. Rozanski and E. Woods, *Software systems architecture: working with stakeholders using viewpoints and perspectives*, 1st ed. Addison-Wesley Professional, 2005.

[2] P. Deutsch, “Fallacies of Distributed Computing - Wikipedia, the free encyclopedia,” *en.wikipedia.org*. [Online]. Available: http://en.wikipedia.org/wiki/Fallacies_of_Distributed_Computing. [Accessed: 14-Apr-2014].

[3] E. Evans, *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[4] I. Alexander, “Stakeholders: who is your system for?,” *Computing and Control Engineering*, 2003.

[5] A. Cockburn, “*Hexagonal Architecture: Ports and Adapters (‘Object Structural’)*,” *alistair.cockburn.us*. [Online]. Available: http://alistair.cockburn.us/Hexagonal+architecture. [Accessed: 15-Apr-2014].

[6] P. Coad, J. Luca, and E. Lefebvre, *Java Modeling Color with Uml: Enterprise Components and Process*. Prentice Hall, 1999.

[7] J. Garrido, “Introduction to Elementary Computational Modeling: Essential Concepts, Principles, and Problem Solving,” *Introduction to Elementary Computational Modeling: Essential Concepts, Principles, and Problem Solving*, Oct. 2011.

[8] V. Law, C. Knox, Y. Djoumbou, T. Jewison, A. Guo, Y. Liu, A. Maciejewski, D. Arndt, M. Wilson, V. Neveu, A. Tang, G. Gabriel, C. Ly, S. Adamjee, Z. T. Dame, B. Han, Y. ZHOU, and D. S. Wishart, “DrugBank 4.0: shedding new light on drug metabolism.,” *Nucleic acids research*, pp. 1091–1097, 2014.

Appendix A – Formal scalability analysis in the p-medicine platform

The p-medicine platform is comprised by a set of state-of-the-art tools and services for the managing, exploration and analysis of biomedical data. Depending on the specific data repositories and the number of concurrent accessions, these tools must handle immense volumes of data, leading to high response times and/or low throughput values. This section provides a thorough analysis of the requirements for ensuring scalability of the developed tools and services.

The report begins with an estimation of the current and near-future needs of the p-medicine platform (i.e. number of simultaneous users that should be supported, number of biomedical databases to integrate and frequency of updates). The report continues with the identification of the p-medicine tools and components that must handle a significant load of work under specific load conditions. This allows discarding several tools and focusing the analysis on those which might become a bottleneck of the platform. Finally, the report concludes with specific scalability analysis for the selected tools and services, performing different benchmarks on each of them.

A.1 Current and near future platform requirements

The platform requirements are estimated in two dimensions: i) Amount of resources (Clinical Trials, biomedical databases) incorporated in the platform by unit of time, and ii) Number of users accessing those resources by unit of time. Regarding the former, the number of ongoing clinical trials in Europe is nowadays almost 16,000, as stated at https://www.clinicaltrialsregister.eu/. The number of available biomedical databases is above 1,500 [A1]. Considering that a single deployment of the p-medicine platform acquires 10% of these resources, and assuming an average size of 1 million items per CT or database and an average of two updates per month (both estimates can be considered as "above" average requirements, since many of the existing clinical trials and databases do not offer public data, and many existing databases do not undergo any frequent updates or any updates at all), the platform should be able to handle around 1350 items per second.

Regarding the latter value (number of users accessing the platform), we can use a value of 10,000 (the European Association of Medical Oncology has around 6500 members, and the International Society of Paediatric Oncology lists around 1500). Assuming that each user makes an average of a single query to the platform every 30 seconds, the platform should be able to support over 300 concurrent requests per second[20]. Again, this can be taken as an "above" average estimate, since not every "possible" user is going to use the platform.

A.2 Preliminary analysis of p-medicine tools & services

The scalability study of the p-medicine platform began with a preliminary performance analysis of the tools comprising the platform. This stage served to identify those tools in p-medicine which could affect the scalability of the platform and therefore produce bottlenecks

[20] This "30 seconds" rate of user requests includes user's "think time" (reading the description of what the server replies, submitting the next query, etc.) and also the response time for the submitted query. The majority of those requests should complete in a time below 1 sec in order for the system to appear responsive to the user, which means that in a half of minute we can get at least 30 requests (from 30 users, because each user makes one request per 30 seconds, and assuming a uniform distribution) with no overlap. So for all the 10,000 users, with the uniform distribution in a 30 seconds time window, we will get at most 10000 / 30 ≈ **334** concurrent requests per second from the corresponding number of users.

This project has received funding from the European Union's Seventh Programme for research, technological development and demonstration under grant agreement No 270089.

in future deployments of the developed technology. As a result, two differential tool groups were generated: i) a group of tools that produced negligible response time values, and ii) a group of tools that presented significant response times and high dependency on the input data size.

A.2.1 Procedure description

The preliminary analysis consisted on shallow tests to detect intensive computing tasks. The tested tools and services were the following:

- Portal
- Ontology Annotator
- ObTiMA
- Data Warehouse (deployed in PSNC infrastructure)
- Data Translator
- Workbench
- Cloud Storage System
- CATS

For each tool, basic tests were run (one simultaneous user). The tests were run manually and response times and/or throughputs were measured. In case of the Portal, the Ontology Annotator and ObTiMA, the tests were performed by manually accessing the tools through a web browser and manually accessing the services provided by those tools.

For the Portal, the tested operations included: opening the main page, log in, selecting the “Documents and Media” page, selecting a document in the Portal, downloading a document, switching between communities, opening the control panel, opening a user profile page, opening the p-medicine tools page.

For the Ontology Annotator, the tests were: opening the available annotation projects page, creating a new project, opening an existing project, selecting HDOT modules, searching for an HDOT class, adding classes to the HDOT window, creating an annotation entry and uploading an annotation to the Data Warehouse.

For ObTiMA, the tests consisted of creating a new trial, entering trial data and uploading the trial data to the Data Warehouse.

The rest of tools were tested programmatically. The tests included the handling the data of a medium-size database (500 patients, 10 attributes for each patient) in case of the Data Warehouse, the Data Translator and CATS, and handling the sequential registry and retrieval of 20 tools in case of the Workbench.

A.2.2 Analysis results

The response time operations performed on the Portal provided negligible values. Similarly, the operations performed in the Ontology Annotator showed good response times (operations completed in less than a second, except for the creation of new projects, which took an average of 5 seconds). The Workbench also provided good response times of less than a second. The rest of tools provided response times of several seconds for most of the tested operations.

The performance of the Portal is acceptable for a web application. While this tool was tested with one single user, the performance hit with more concurrent users should not be elevated as it would be handled by the container hosting it. In case of the Ontology Annotator, the response times are also above acceptable for a web application and the scalability with users should depend on the application container on which this tool runs. The percentage of users that potentially might access this tool will be very low, and database annotations will be created only once per database. Both these tools can be discarded as possible performance bottlenecks in the p-medicine platform, since their response time does not depend on the volume of data handled, and they are not expected to receive high amounts of requests from users, since they do not handle data directly.

The rest of tools were selected for further scalability analysis. Details are provided in the following section.

A.3 Tool-specific scalability analysis

The advanced scalability analysis of the tools selected in the previous section involved more complex tests. The goal was to measure the effect of the input data size and number of simultaneous requests on the response time and throughput of the tested tools. Each tool analysis was specifically designed to find the capability of scaling up upon the growth of the mentioned input variables.

A.3.1 Data Warehouse

The p-medicine Data Warehouse is a central repository of heterogeneous data. As a service itself, it features a REST-based interface supporting the following operations:

- Add/Retrieve files to/from the *filestore*.
- Add/Retrieve semantic triples extracted from uploaded files to/from the *triplestore*.
- Retrieve DICOM images.

The Data Warehouse serves as a heterogeneous data integration framework, where disparate databases are pushed to the *filestore* and their HDOT-compliant version are pushed to the *triplestore*.

Test description

In order to evaluate the performance of the Data Warehouse, three different tests were designed:

- The first test checks the performance on the *triplestore* when adding/retrieving triples to/from the data warehouse.
- The second test checks the performance on the *filestore* when uploading/downloading files to/from the data warehouse.
- The third test checks the memory usage for both the *triplestore* and the *filestore*.

The tests were executed with different file sizes containing data about 1, 100, 500, 1000 and 5000 patients (each file contained 10 items per patient, so the item count was, respectively, 10, 1000, 5000, 10000 and 50000 items). All tests included monitoring of processor activity and file system activity. The DICOM server was excluded from the tests, since the workload is expected to be primarily supported by the two other components, which serve for more general purpose tasks.

The configuration employed during the tests relied on the following software:

- Key-Value Database: Redis 2.8.3
- Application Server: Apache Tomcat 7.0.26
- RDF Storage: OpenRDF Sesame 2.7.6

In terms of hardware and operating system, an Intel® Core®2Duo Processor T7700 with 2 Cores, 2 GB RAM and a mechanical hard drive system (7200rpm) running Linux Kernel : 3.2.0-24-virtual 64 Bit running Ubuntu 12.04 (“Precise”) was employed.

Test results

The results obtained in test 1 are shown in figure A1.

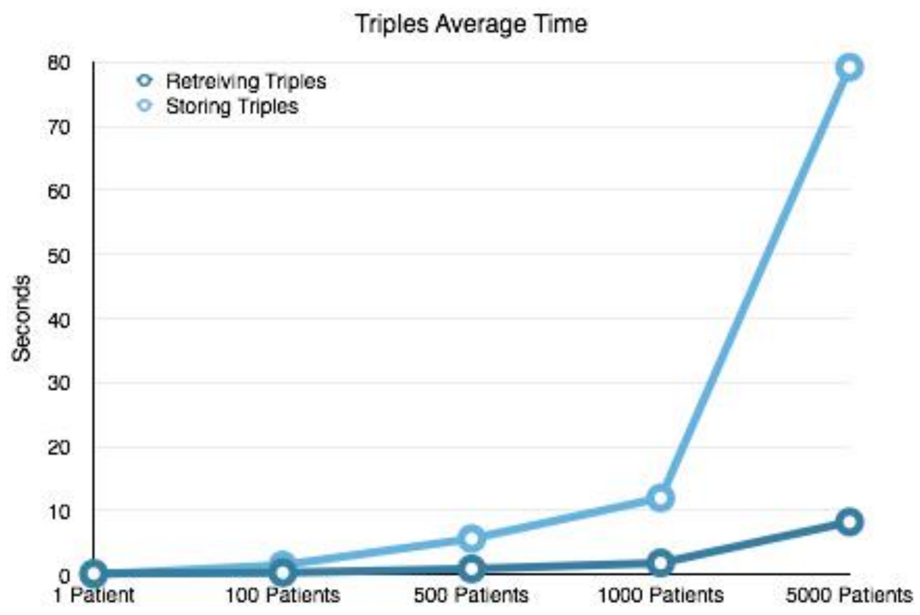


Fig A1: response times for storing and retrieving data in the *triplestore*.

Test 2 results are depicted in figure A2.

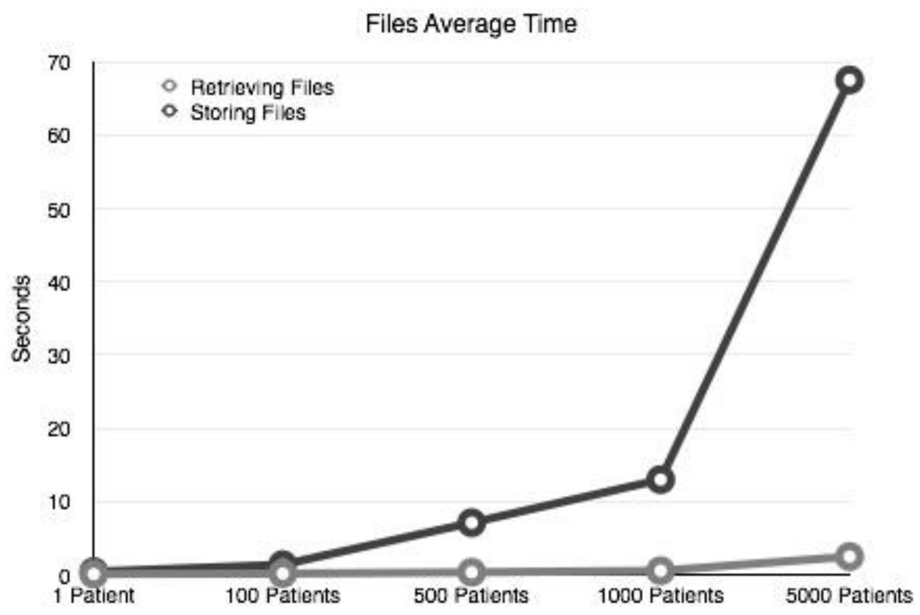


Fig A2: response times for storing and retrieving triples in the *filestore*.

Finally, figure A3 shows the results of the third test.

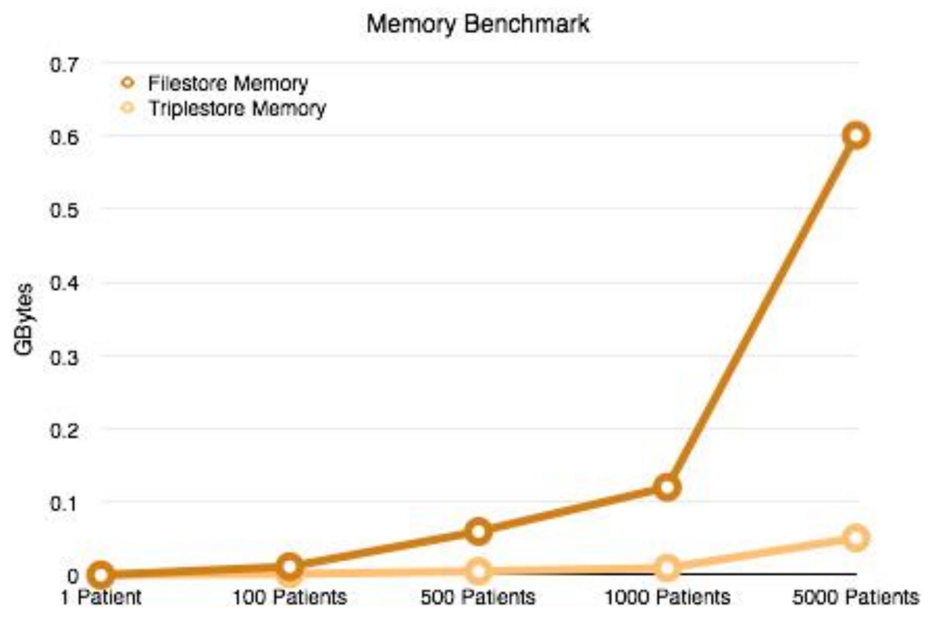


Fig A3: memory requested by the *filestore* and the *triplestore* during the tested operations.

Scalability Analysis

The *triplestore* test shows how the storage of new triples produces much higher response times than the retrieval of triples, with values around 3000 patients per minute, or 500 items per second. The monitoring of the machine running the test showed that the element limiting the performance was the IO system (hard drives).

The results of the *filestore* test show that pushing files into the Data Warehouse takes again much more time than retrieving files. The reason in this case is that, for any uploaded file, there are three additional files created: the one containing the raw triples, the one with the ontology and the one containing metadata. The obtained values are again around the 500 items per second.

With regards to the memory consumption, the third test showed that the filestore comsumes much more memory than the triplestore. In any case, the memory consumption grows almost linearly with the number of data items

The achieved performance of the Data Warehouse is slightly lower than the highest estimated current requirements for data processing performance. However, the tests show that response times can be greatly reduced by using more advanced data storage units, such as SSDs. The memory can also be a limiting parameter for the Data Warehouse. Increasing the amount of RAM will allow the Data Warehouse to scale up in the event of increased data processing requirements.

A.3.2 Data Translator

The Data Translator handles the homogenization of datasets to an HDOT-compliant format. This involves an automatic data analysis and refactoring process, which heavily depends on the size of the managed datasets. The Data Translator is designed to concurrently handle several translations and make use of multi-processor configurations.

The data translation process is guided by the database annotations. A database annotation (represented by an XML file describing semantic relationships between elements of the database and elements of HDOT) allows the Data Translator to automatically perform the homogenization of that database. The annotation can involve one or more elements of the database, depending on what attributes shall be translated. For each element of the annotated database, the corresponding annotation will store one *annotation entry*. Therefore, the size of a database annotation can be regarded as the number of *entries* contained in that annotation. A small-sized annotation can be formed by 1 to 5 *entries*. A medium-size annotation can contain around 15 *entries*. Larger annotations can contain dozens of *entries*.

As a result, the *load of work* of the Data Translator depends on three variables:

- Number of simultaneous requests: the number of simultaneous translations that must be achieved (can be seen as the number of simultaneous users making use of the translation service).
- Size of databases: the number of items that a database contains. In the scalability analysis, we focus on the number of rows of the databases.
- Size of the database annotations: the number of *entries* of the employed database annotations.

Test description

The Data Translator scalability analysis aims to determine how the variables described above affect the throughput of the Data Translator. The analysis also includes the use of different machine configurations (varying in number of processing units and amount of RAM) to measure what resource allows the system scaling up. Two different scalability tests were run on the Data Translator:

- **Test #1**: This test aimed to measure the sensibility of the throughput of the Data Translator upon different input sizes, maintaining a fixed machine configuration. More specifically, we set the number of active processing units to 4, and the amount of used RAM to 4GB. We set up a test with sequential accessions (no simultaneous users) and different values of database size and annotation size. The database size included the values **1**, **100**, **500**, **1000** and **5000** rows, and the annotation sizes were **1**, **2**, **5**, **10**, **20** and **40** *entries*.
- **Test #2**: This test focused on finding the relation between machine parameters (number of active processing units, amount of RAM) and system throughput. In this case, the input size was maintained constant (1000 rows, 20 *entries*), but the number of simultaneous accessions varied from **1** to **64** concurrent accessions. The RAM amount varied: **256MB**, **512MB**, **1GB**, **2GB** and **4GB**. For each of these values, the system was tested with **1**, **2**, **3** and **4** active processing units and **1**, **2**, **4**, **8**, **16**, **32** and **64** concurrent accessions.

The tests were performed with a dedicated Corei7 860 system (4 cores) including 8GB of RAM. The machine used a Windows Server 2008 operating system and the Java 7 update 51 runtime environment.

Test results

The results for test #1 are shown in table A1 (times are given in seconds).

# Rows	**# Entries**						
	1	**2**	**5**	**10**	**20**	**30**	**40**
1	1,29	1,76	1,78	1,17	1,72	2,06	2,01
100	1,89	1,89	2,22	2,20	1,91	1,96	1,47
500	2,53	2,85	2,68	2,10	3,55	2,97	3,20
1000	3,30	3,49	4,43	4,14	3,29	4,44	4,03

5000	11,07	9,48	10,43	11,21	12,14	14,01	16,52

Table A1: response times obtained for test# 1.

Figure A4 shows these values graphically.

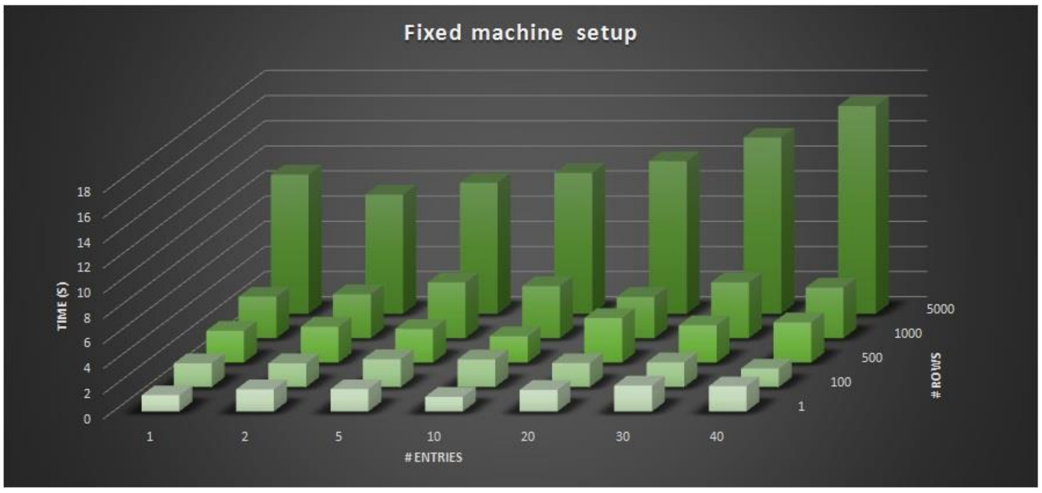


Fig A4: the response times obtained for the test# 1.

For the second test, the results for the different amounts of RAM are shown in tables A2, A3, A4, A5 and A6.

#Accessions	# Processing Units			
	1	**2**	**3**	**4**
1	4,75	4,01	4,43	3,88
2	6,73	5,52	5,64	5,80
4	12,02	10,16	9,77	10,83
8	35,39	-	-	-
16	-	-	-	-
32	-	-	-	-
64	-	-	-	-

Table A2: response times for test# 2 run with 256MB of RAM

#Accessions	# Processing Units 1	2	3	4
1	4,23	3,63	4,02	3,93
2	6,79	6,05	5,50	5,31
4	12,86	10,61	9,12	9,66
8	22,91	21,12	17,64	18,24
16	49,00	45,08	40,01	40,72
32	-	-	-	-
64	-	-	-	-

Table A3: response times for test# 2 run with 512MB of RAM

#Accessions	# Processing Units 1	2	3	4
1	4,16	4,11	3,44	3,63
2	6,66	5,09	5,59	5,32
4	12,48	10,14	8,12	8,64
8	23,59	20,66	16,84	17,12
16	44,66	40,77	33,52	32,33
32	96,63	84,12	71,13	72,93
64	-	-	-	-

Table A4: response times for test# 2 run with 1024MB of RAM

#Accessions	# Processing Units 1	2	3	4
1	4,28	3,84	3,83	3,38
2	6,26	6,35	5,31	5,08
4	12,68	11,04	8,38	7,71
8	23,38	18,71	15,22	15,06
16	46,56	36,52	31,49	32,75

32	92,30	75,81	64,44	64,14
64	196,55	167,72	139,01	137,37

Table A5: response times for test# 2 run with 2048MB of RAM

#Accessions	# Processing Units			
	1	2	3	4
1	4,47	3,52	3,70	3,62
2	6,03	5,76	4,88	4,86
4	12,14	9,37	8,11	9,59
8	23,57	17,27	16,22	16,49
16	47,21	33,95	30,57	30,84
32	95,09	69.65	62,74	63,49
64	185,42	146,77	132,44	127,57

Table A6: response times for test# 2 run with 4096MB of RAM

Figures A5, A6, A7, A8 and A9 graphically represent these values.

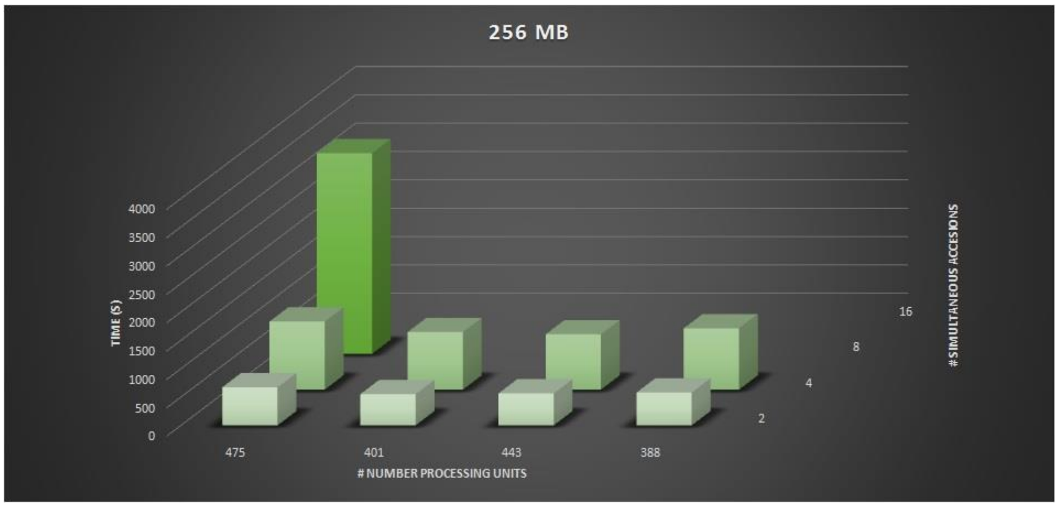


Fig A5: the response times obtained for the test# 2, with 256MB or RAM.

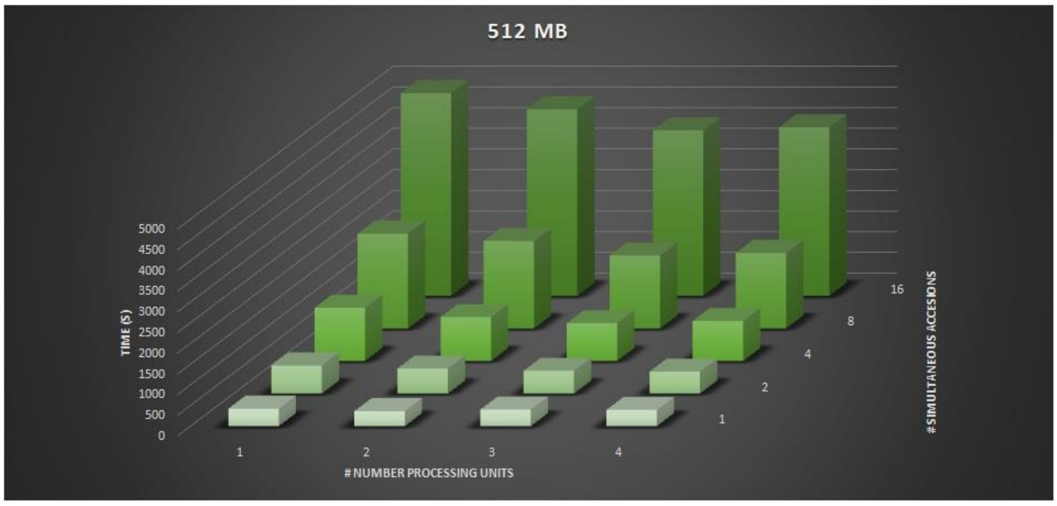


Fig A6: the response times obtained for the test# 2, with 512MB or RAM.

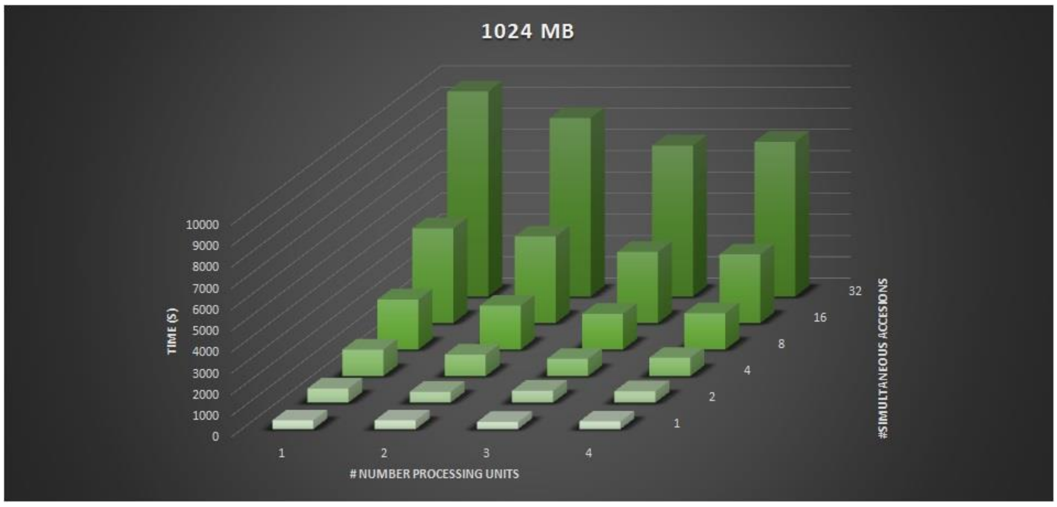


Fig A7: the response times obtained for the test# 2, with 1024MB or RAM.

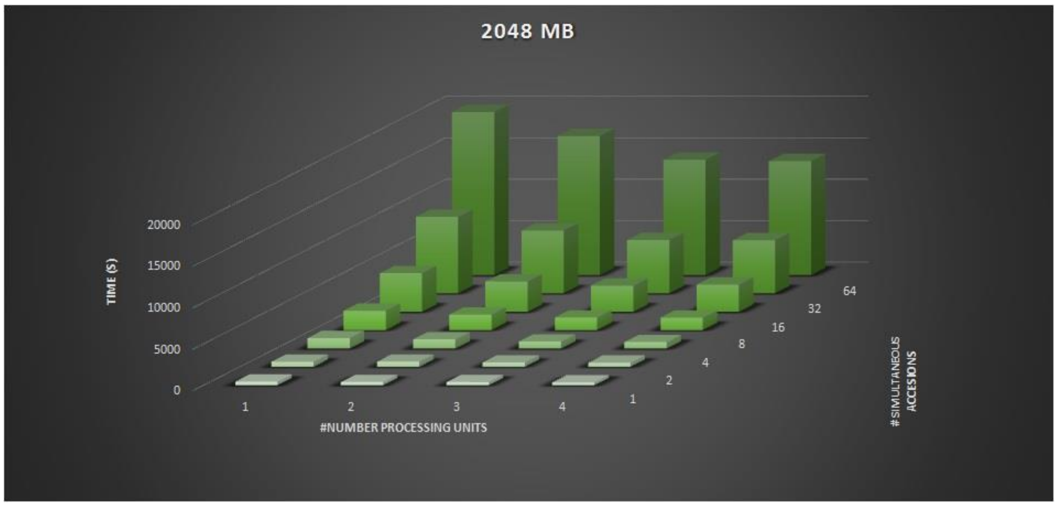


Fig A8: the response times obtained for the test# 2, with 2048MB or RAM.

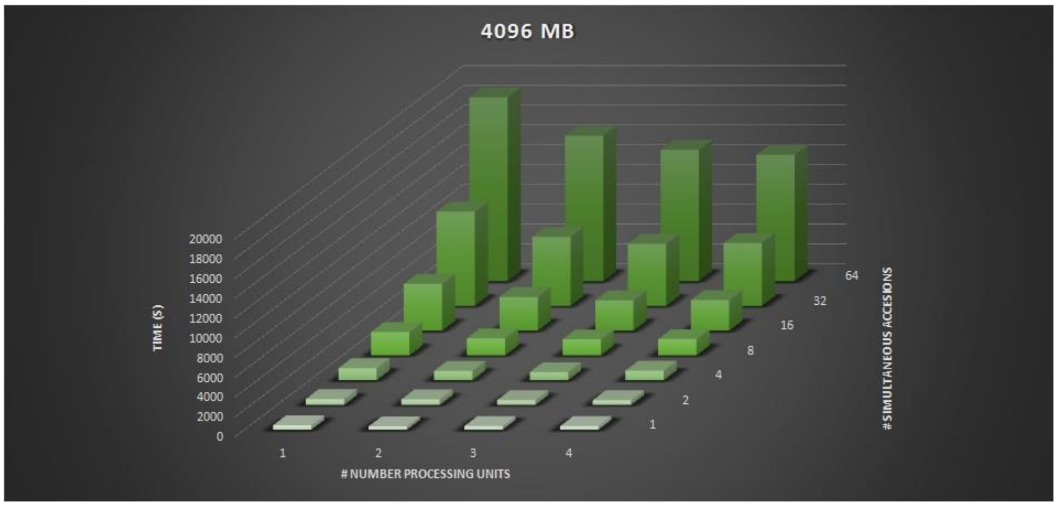


Fig A9: the response times obtained for the test# 2, with 4096MB or RAM.

<u>Scalability Analysis</u>

Test #1 shows a clear relation between the system throughput and input data size. However, the parallelized algorithm allows to maintain that dependency below linear. For example, with a fixed value or 5000 rows, moving from 20 entries to 40 entries increases the response time 1,36 times. With a fixed value of 40 entries, moving from 500 rows to 5000 rows increases the response time 5,61 times. These data show that, in principle, the system is able to scale upon the growth of the input data size, both in terms of database rows and annotation entries.

The results of test #2 show that the number of simultaneous accessions directly affects the performance of the system, and doubling the amount of accessions approximately reduces performance by a 50%. For each fixed RAM configuration, there is an amount of simultaneous accessions where this linear dependency breaks, and response time start rising more abruptly. In the case of 256 MB of RAM, for a value of accessions between 4 and 8 the system stalls and performance greatly degrades (even makes the system fail). For 512 MB of RAM, this value lies between 8 and 16 accessions. For 1 GB of RAM, it is between 16 and 32 accessions. These data indicate that there is a linear dependency between the amount of system RAM and the maximum amount of allowed simultaneous accessions (for the system to scale). With respect to the number of processing units, the data show that they allow reducing response times significantly, but they do not specifically allow the system to scale up upon large input sizes.

A.3.3 Workbench

The Workbench is basically a registry/repository for Tools and domain specific Services. As a service itself, it features a REST-based interface supporting the following:

- Full Text Search for textual attributes (names, descriptions, tags, etc.)
- Combined queries, e.g. search the names/descriptions and tags of tools in a specific category: **`../v1?q=rna&qtag[]=sequence&qcat=Bioinformatics`**
- Both retrieve and update/store interfaces is used by the workbench portlet in the Portal

Input data size in this case is measured by the amount of tools registered in the portal.

<u>Test description</u>

The setup for the stress tests is shown in the figure on the right. The benchmarking client is the Apache JMeter[21] running in a Windows 7 machine that accesses the workbench application using a typical 10/100Mbps local area network. The communication between the client and the server is based on the HTTP/REST interface using JSON messages while the database connections are using the PostgreSQL JDBC 4 driver[22].

The database contains around 500 "tools" that we consider more or less an example of a typical load. The tools have been annotated with 1000 (free text) "tags" and 1300 "semantic terms" from the EDAM ontology[23] and have been classified in 10 categories.

We consider two basic scenarios:

- Test #1: a client application representing a principal user submits queries in order to locate tools matching some given description and tags
- Test #2: a synthetic scenario where the searching for tools happens with concurrent updates to the tools' tags. 90% of users perform searches, while 10% perform updates

Additionally we incorporate some randomization in order to increase confidence in the results:

- 10 different search queries are randomly sampled to select one before each request
- 5 different "tag updates" actions randomly sampled to select one before each request

In the second scenario the client chooses in (approximately) 90% of time to do one of the search queries and 10% of time to do one of the tag updates

We perform stress testing by varying the number of concurrent users: From 100, 200, 500, 800, to 1000. Each user makes 5 requests and each scenario is repeated 20 times. Therefore, we make:

- 100 * 5 * 20 = 10,000 total requests for the 100 users
- 200 * 5 * 20 = 20,000 total requests for the 200 users
- 500 * 5 * 20 = 50,000 total requests for the 500 users
- 800 * 5 * 20 = 80,000 total requests for the 800 users
- 1000 * 5 * 20 = 100,000 total requests for the 1000 users

Finally we make the following measurements:

- Throughput, i.e. the number of successful requests per second
- Response times, i.e. the time taken to receive a complete response to a submitted request. For this we compute some statistical metrics for the "central tendencies" (mean, median) and the "Dispersion" (standard deviations, IQRs, range (max - min)) of measurements.

In terms of the software technologies used, the following applies:

- Relational Database: PostgreSQL 9.3
- Application Server: Clojure (JVM "1.6.0_33" Hotspot Server), with embedded Jetty[24]. The application uses a (conservative) "ThreadPool" of 30 threads, each holding a single database connection (Therefore there are 30 concurrent connections to the database)

In terms of the hardware and operating system used in the stress tests:

[21] https://jmeter.apache.org/

[22] http://jdbc.postgresql.org/

[23] http://edamontology.org/

[24] http://www.eclipse.org/jetty/

- 2 Intel® Xeon® Processors E5-2630 with 6 Cores each yielding 12 Cores. With 2 "hyper" threads, the system appears to provide 24 threads in total.
- 64 GB RAM
- Linux Kernel : 3.2.0-37-generic SMP 64 Bit running Ubuntu 12.04 (“Precise”).

This machine runs both the Database and the Application Server but its processing power is shared with other services (e.g. a DICOM server) used internally at FORTH.

Test results

Figure A10 shows the results obtained for the first test.

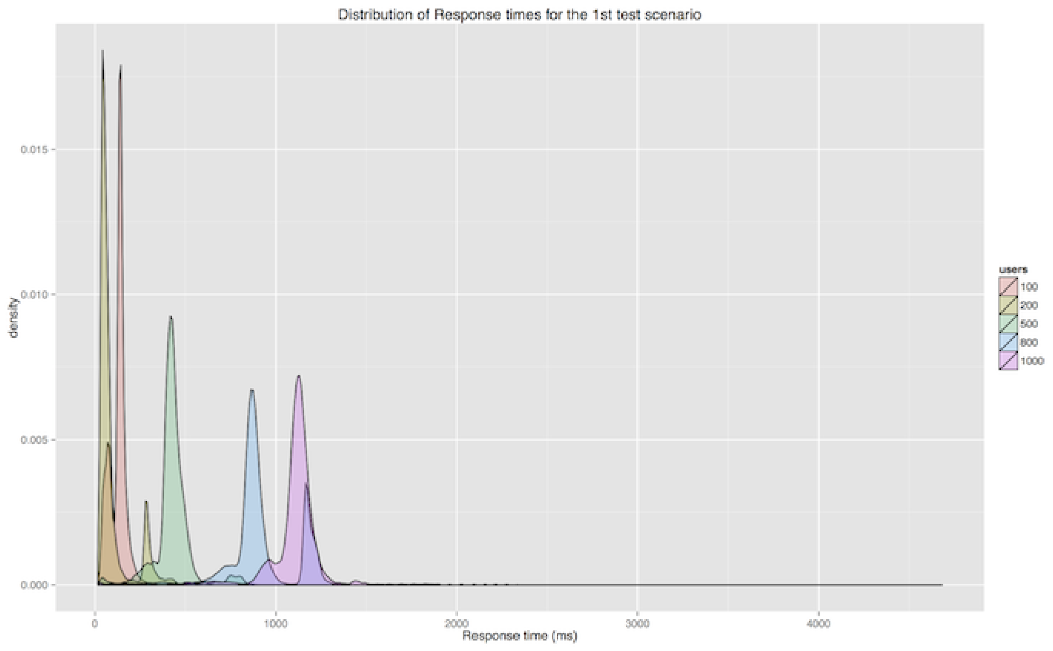


Fig A10: Distribution of response times for the performed requests in milliseconds. The highest response times are slightly above 1 second.

Table A7 shows a summary of these results, providing several statistical parameters.

Users	Min	Max	Mean	Median	IQR	1st quartile	3rd quartile	90th percentile
100	20	551	127.15	137	51.25	96.75	148	169
200	17	755	96.18	59	42.00	44	86	282
500	18	968	429.88	426	64.00	398	462	508
800	16	4255	914.97	882	121.00	844	965	1186
1000	17	4681	1105.70	1123	78.00	1083	1161	1207

Table A7: statistical parameters obtained in the first test (milliseconds).

Throughput was also measured for the first test. Throughput results are provided in figure A11.

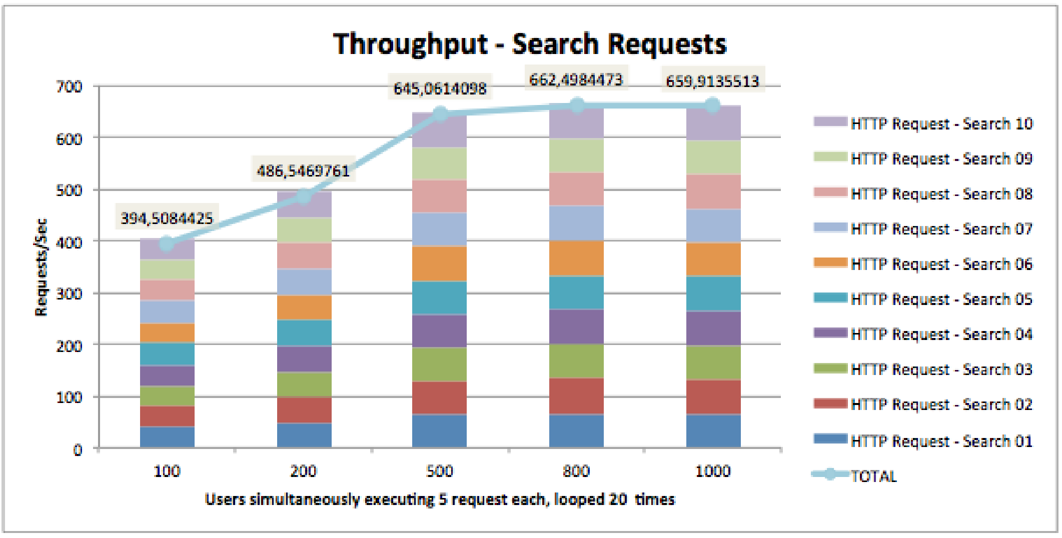


Fig A11: obtained throughput (search requests completed per second) for different amount of simultaneous users for test #1.

For the test #2, Figure A12 shows the obtained results.

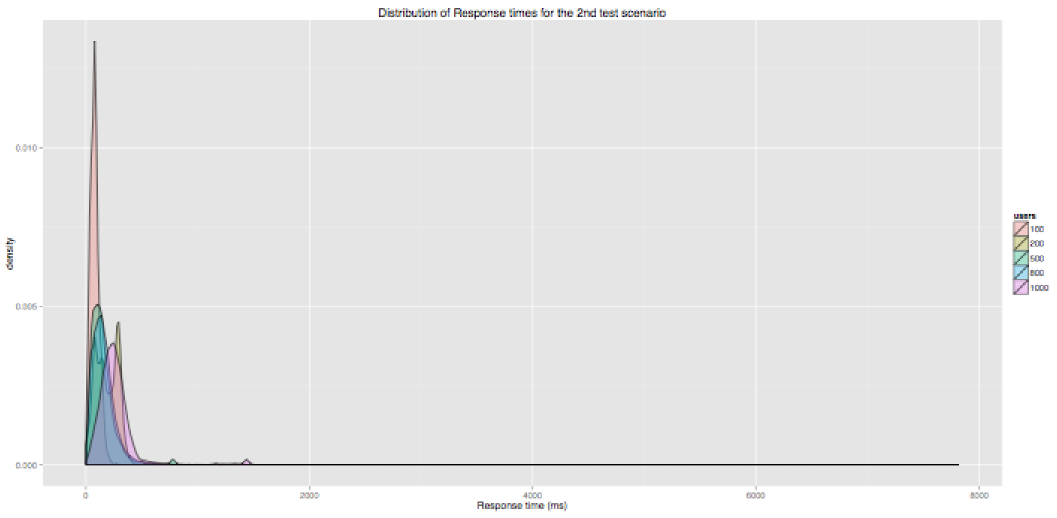


Fig A12: distribution of response times in milliseconds for the second test. Response times are lower compared to test #1, staying below 1 second

Table A8 provides the statistical parameters extracted for this test.

Users	Min	Max	Mean	Median	IQR	1st quartile	3rd quartile	90th percentile
100	3	440	78.96	76	51	49	100	132
200	3	530	186.64	180	177	100	277	308
500	2	947	151.35	128	112	78	190	270
800	2	3352	155.72	141	117	87	204	275
1000	2	7813	270.23	240	141	171	312	389

Table A8: statistical parameters obtained in the first test (values represent milliseconds).

Throughput was measures again for this test, as shown in figure A13.

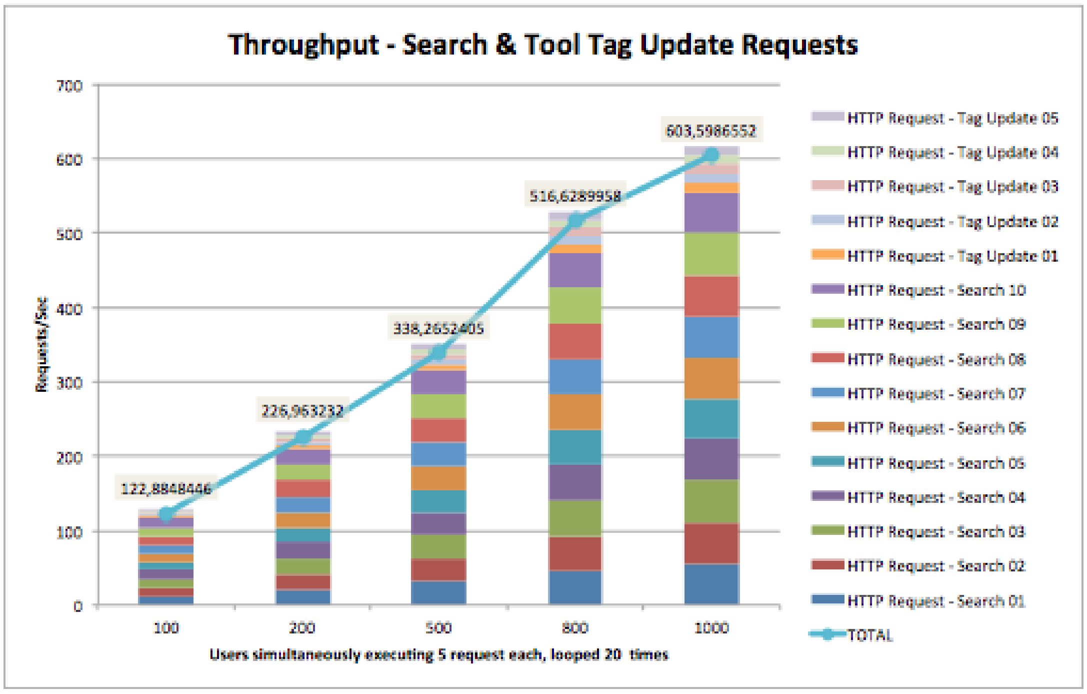


Fig A13: obtained throughput (search requests completed per second) for different amount of simultaneous users for test #2.

Scalability Analysis

We have made some extensive testing by taking into account the number of concurrent "users" of the system. The results show that the Workbench is able to achieve around 600 requests per second in synthetic load even with 1000 simultaneous users, which is more than enough for the current needs.

In the case that more raw performance is required, a more complex application architecture is needed. In particular since the contents of the tool repository rarely change, the system's performance can be greatly increased by the introduction of a “caching” layer. Moreover, again due to the large ratio of read to write operations, replicas of the tool repository can be used to achieve horizontal scaling.

A.3.4 Cloud Storage System

P-medicine Cloud Storage System is the lowest level component in the data management architecture of p-medicine. It provides REST interfaces for managing the storage of files in the cloud environment and is built based on OpenStack technology.

The Cloud Storage System is used by Data Warehouse as a storage backend for files and is also linked to DICOM server for storing medical images. It relies on the OpenStack Swift software[25]. It’s functionality is an archive backend for patient data. OpenStack Swift is responsible for replication of the data distributed across multiple drives in the server cluster. All the data is accessible through a RESTfull API. Main OpenStack Swift operations are to upload, download and remove data from the system.

Test description

The performed tests measured the transfer speeds to and from the Cloud Storage System. The tests consisted in uploading and downloading data files of different sizes (10, 50, 100

[25] https://swiftstack.com/openstack-swift/

and 500 Mbytes), and measuring the speed. Measurements were made using 40 files of each size, and final speed for each file size was calculated as the average of the 40 runs.

In case of the operation of file deletion, the amount of deleted files was also considered as a measurement parameter.

The last test focuses on the speed of creating and removing containers for files. In this case, the number of creaion and removal operations ranged from 5 to 200.

All tests were run on a system containing one proxy server and five storage servers. All servers machine work on AMD Opteron 2,6Ghz (6 core) processors. The proxy server uses 2GB of RAM and the storage servers use 1GB of RAM.

Test results

The first test measured the average download transfer rate of the Cloud Storage System. Figure A14 shows the results of this test.

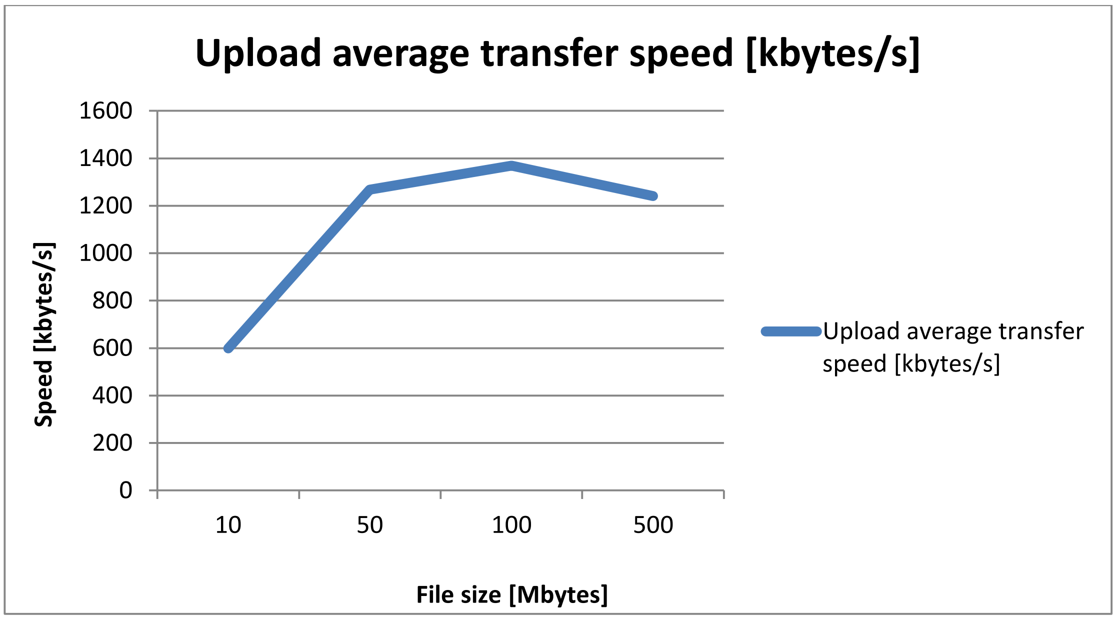


Fig A14: Average download speed achieved by the Cloud Storage System.

Table A9 provides the values obtained in this test.

File size [MBytes]	10	50	100	500
Upload average transfer speed [kBytes/s]	598,29	1267,90	1369,74	1241,00

Table A9: The values obtained for the first test.

The second test measured the download transfer speed for the same files. Figure A15 shows the results.

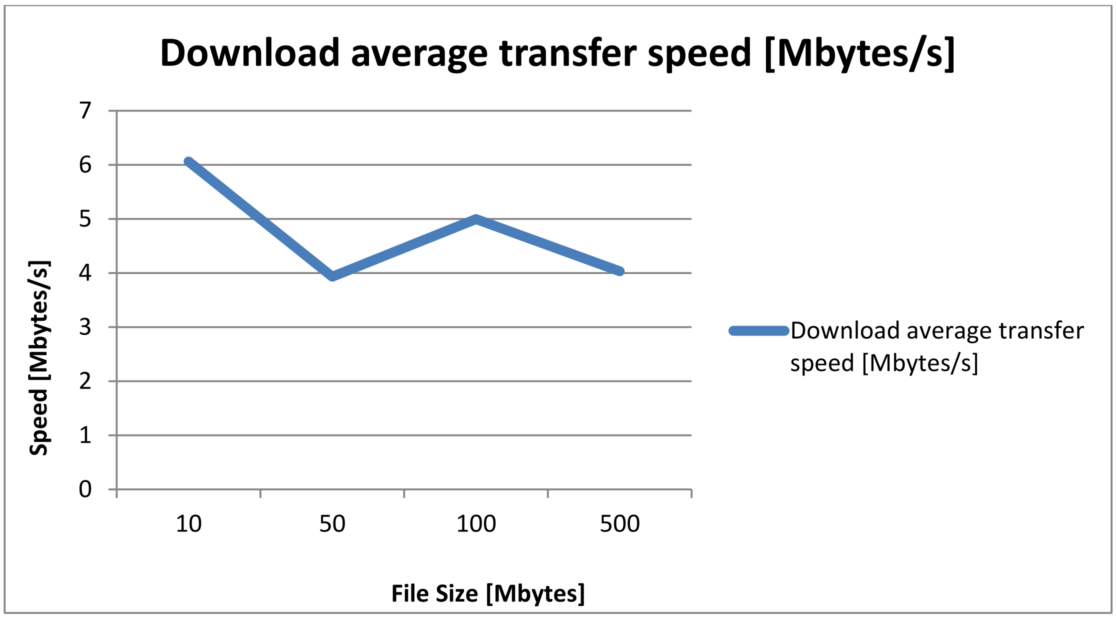


Fig A15: Average upload speed achieved by the Cloud Storage System.

And table A10 provides the values for this test.

File size [Mbytes]	10	50	100	500
Download average transfer speed [Mbytes/s]	6,06	3,94	5,00	4,03

Table A10: The values obtained for the second test.

The third test measured the file removal response time (the time in seconds to delete certain amount of files). Figura A16 shows the obtained results.

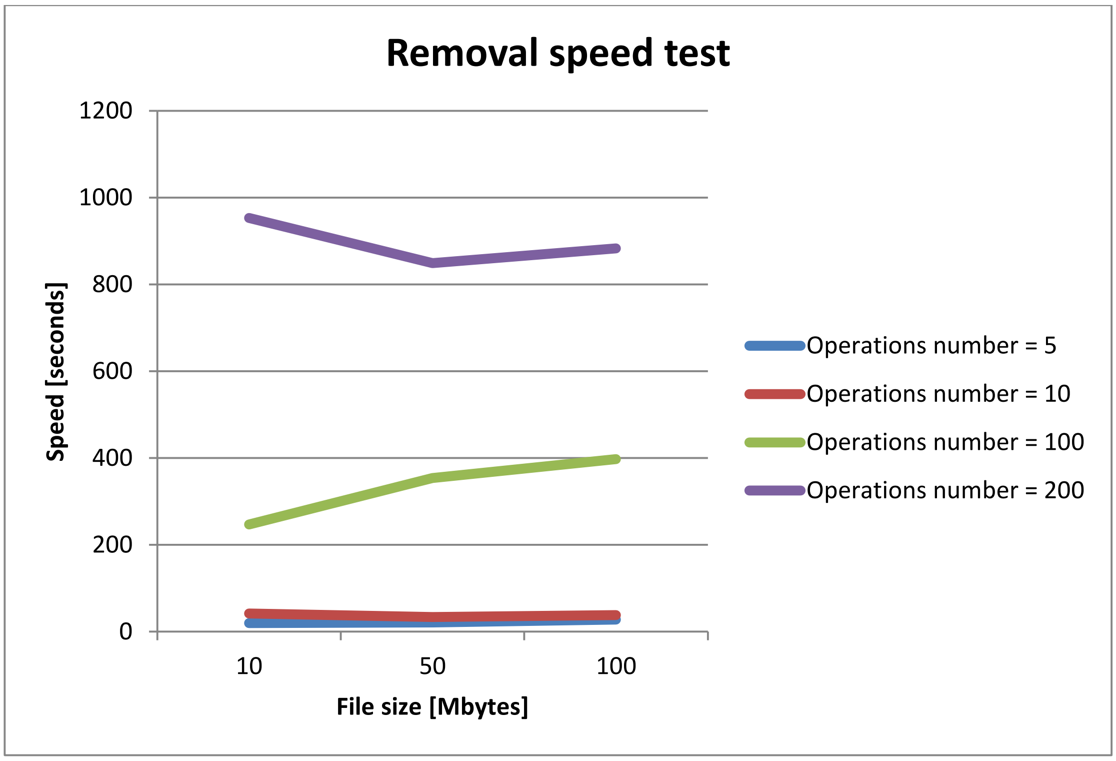


Fig A16: Average response time in seconds for the operation of deletion of files.

Scalability Analysis

The performed tests show that the Cloud Storage System performance varies depending on several factors. In case of upload operations, the system scales worse with small files (less than 50Mbytes), and reaches speeds of around 1200Kbytes per second with large files. In the case of download operations, obtained speeds are much higher, with values around 5Mbytes per second (and slight decreases for larger files). Finally, deletion operations do not show a dependance on file size, and the system is able to delete around 0,2 files per second.

The uploading speeds obtained by the current configuration of the Cloud Storage System should be capable of handling current requirements. By average, one database item is smaller than 1Kbyte in size, so the system is able to process the maximum estimated data volumes. The obtained download speeds are more than 1000 times greater than the upload speeds. Current needs are perfectly fulfilled by this capability. In terms of file deletion, this is a rarely required operation, so the obtained values are more than enough to satisfy the needs of the platform.

Proxy services depend on number of nodes and speed of networking for better performance. The system can be easily scaled up by adding more nodes and providing faster networking infrastructure. In extreme situations, more than one proxy can be used for performing load balancing, and providing larger amounts of RAM to each proxy will enable the system to maintain performance under heavy traffic. Servers responsible for storing data do not need as much RAM or CPU as the proxies, but their performance can be scaled up by providing high-speed storage units, such as SAS or SSD based units.

A.3.5 CATS

The p-medicine pseudonymisation services are implemented by CATS (Custodix Anonymisation Tool Services). CATS provides patient and medical data pseudonymisation through a REST-based web service. Data is submitted to CATS, where it will be processed asynchronously one by one sequentially to remove all identifiers. Subsequently the processed data is sent to the data warehouse where it will be stored for further use.

Test description

The test starts after the files containing the data are uploaded to CATS. These files are processed by the tool to remove any information providing patient identification. The test measure the response time of this process, and ends before the data is submitted to the data warehouse.

A submitted REST message can contain data elements from multiple patients to be processed. The amount of patients is varied, and the time it takes for each request to be processed is recorded. These measurements will provide us with the necessary information to assess the scalability of the CATS service and give us an insight into the average response time for the CATS service.

For the performed test the amount of patients ranges from 1 to 5000. The test makes use of a public dataset containing miRNA expression data and used within p-medicine.

The tests were performed on a virtual machine with 2 Intel cores running at 2,93 GHz and 1,5 GB of RAM running CentOS.

Test results

The results for the tests are shown in table A11, with the amount of patients and data size given, and the time to process measured.

#Patients	Size (kb)	Time (s)
1	21	17
5	61	24
50	504	16
100	997	23
500	4939	49
1000	9868	95
3000	29584	305
5000	49291	445

Table A11: processing response times obtained by CATS for different file sizes

The relation between processing time and file size is shown in figure A17.

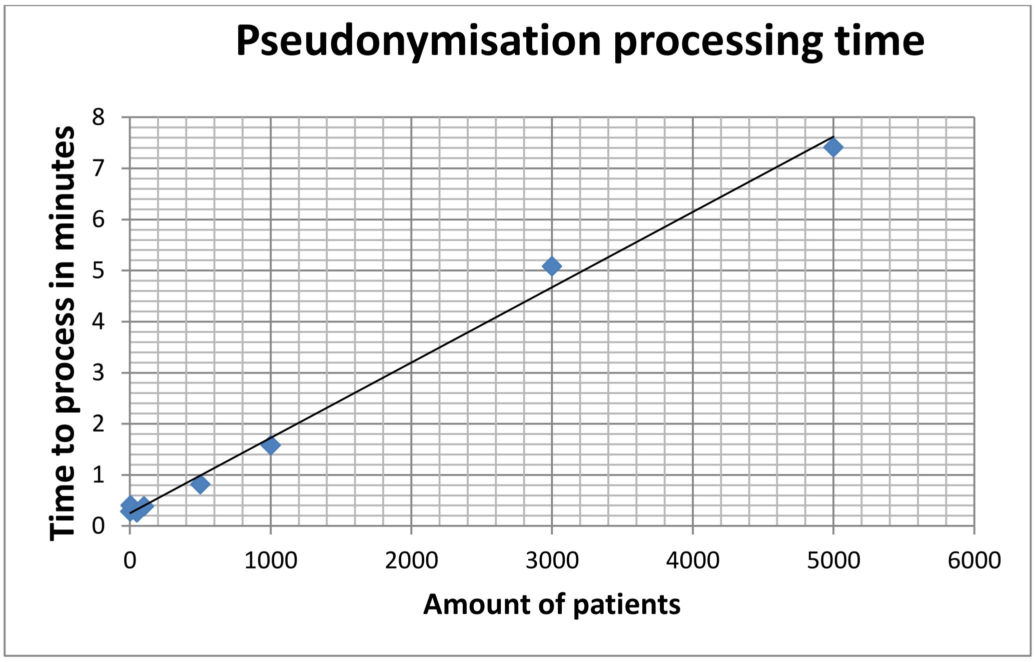


Fig A17: relation between the amount of patients (file size) and processing time in CATS

Scalability Analysis

The graph shows that the time to process grows linearly with the amount of patients. This is certainly the case for larger patient amounts. For smaller amounts this linear dependency is less explicit. The reason is that pseudonymisation requests are put in a waiting line that is polled. For smaller data sizes the time it takes to process a file is much shorter than the polling frequency, so the time it takes to process small data sizes is almost independent from the amount of patients.

The processing time per patient, which is the slope of the curve, is shown to be almost constant once the amount of patients grows beyond 500, reaching a value around 10 patients per second. It should be noted however that this processing time is dependent on the contents of the data and the pseudonymisation operations that should be performed.

The CATS service does not need to process every database item, but only those that can lead to patient identification. The performance requirements are therefore not as high as other data processing tools, such as the Data Translator. Moreover, not every dataset to be integrated in p-medicine stores patient-related data, so CATS does not need to process every possible dataset. Our preliminary estimations indicate that the achieved performance of CATS is nearly enough to cope with the needs of the p-medicine platform. In any case, the data processing carried out by CATS is able to benefit from multiple node configurations, thus allowing the tool to scale up upon large volumes of data.

A.4 Conclusions

The scalability analysis carried out on the different tools and services of the p-medicine platform allowed to evaluate the capability of this platform to handle the amounts of data and user requests. Although some more testing has to be carried out the initial results show that the tools comprising the platform are able to cope with current estimated performance requirements, as tested in local configurations. Furthermore, specific requirements have been identified for ensuring the scalability of each tool, allowing producing technical guidelines for future deployments of the platform.

The scalability proof of the different tools is also a proof of the technical viability of the platform architecture approach, in terms of handling of data and user requests. The service oriented architecture followed in p-medicine allows adapting the platform to individualized needs, and facilitates its long-term sustainability. As each tool and service is highly decoupled from each other, resources can be easily redistributed and reconfigured.

Scalability of the developed tools is a primary concern in p-medicine, together with clinical validity and sustainability. The executed tests allow to state, with a high degree of confidence, that the platform can scale up and maintain good performance upon future needs.

A.5 References:

[A1] Fernández-Suárez XM, Rigden DJ, Galperin MY. The 2014 Nucleic Acids Research Database Issue and an updated NAR online Molecular Biology Database Collection. Nucleic Acids Res. 2014 Jan;42(Database issue):D1-6.

Appendix B - Abbreviations and acronyms

ABAC	Attribute Based Access Control
CDS	Clinical Decision Support
DDD	Domain Driven Design
EHR	Electronic Health Record
HDOT	Health Data Ontology Trunk
IdP	Identity Provider
IT	Information Technology
JSON	JavaScript Object Notation
PHR	Personal (or Patient) Health Record
REST	Representational state transfer
SAML	Security Assertion Markup Language
UI	User Interface
VPH	Virtual Physiological Human
XACML	Extensible Access Control Markup Language