MICKAËL MISBACH MASTER THESIS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
LABORATY FOR COMMUNICATIONS AND APPLICATIONS 1
&
THE HYVE

Building a common and privacy-preserving front end for open-source clinical research platforms

The Hyve Supervisors:
Ward WEISTRA*
Dr. Bo GAO*

Master Student:
Mickaël MISBACH*†

EPFL Professor:
Prof. Jean-Pierre HUBAUX†

EPFL Supervisors:
Dr. Jean-Louis RAISARO†
Dr. Juan TRONCOSO-PASTORIZA†

†: *{firstname.lastname}@epfl.ch*
∗: *{firstname}@thehyve.nl*

Abstract

Being able to exploit large and heterogeneous medical data is crucial for realizing the promise of precision medicine to its full potential. Yet, currently, due to the presence of multiple and fragmented systems at different clinical sites, it is often difficult to enable researchers to access the data they need. Privacy and security concerns also represent major obstacles that need to be overcome in order to provide access to sensitive medical data that are usually not exposed by clinical sites for the fear of data leaks.

To address these challenges, we propose a system that employs Glowing Bear as a common front end to the widespread clinical research platforms i2b2, tranSMART and MedCo. Our proposed system uses IRCT, the official implementation of the PIC-SURE API, which acts as an interoperability layer that translates clinical research queries into native API languages. With our work, we take a step towards the technical convergence of i2b2 and tranSMART, giving clinical sites the means to share their data more easily. Even more, with the support of MedCo, a cohort explorer with strong privacy and security guarantees based on federated i2b2 instances, we enable clinical sites to share sensitive data that would be difficult to share otherwise.

Contents

1 Introduction **4**

1.1 Objectives 5

1.2 Solution Requirements 6

1.3 Outline 6

2 Background & Related Work **7**

2.1 Background 7

2.1.1 i2b2 7

2.1.2 tranSMART 8

2.1.3 PIC-SURE API 8

2.1.4 Glowing Bear 9

2.1.5 OpenID Connect 9

2.1.6 MedCo 11

2.2 Related Work 11

3 System Architecture **12**

3.1 Choice of Open-Source Technologies 12

3.1.1 Clinical Research Platforms: i2b2, tranSMART and MedCo 12

3.1.2 Front End System: Cohort Explorers 14

3.1.3 Identity Provider 14

3.2 Building the System Architecture . . . 15

3.2.1 Assembling the Building Blocks . . . 15

3.2.2 Technical Choices Summary . . . 16

3.2.3 System Overview . . . 16

4 Interoperability Layer for Clinical Research Systems 18

4.1 Detailed Workflow . . . 18

4.2 Identity and Access Management . . . 20

4.2.1 Authentication . . . 20

4.2.2 Authorization . . . 21

4.2.3 Implementation . . . 22

4.3 Front End: Glowing Bear . . . 23

4.3.1 Initialization . . . 23

4.3.2 Query Terms Tree . . . 24

4.3.3 Query Construction . . . 25

4.3.4 Query Request & Result Retrieval . . . 28

4.4 PIC-SURE to i2b2 API Translation . . . 30

4.4.1 Query Terms Tree . . . 30

4.4.2 Query Translation . . . 31

4.4.3 Results Management . . . 33

4.4.4 Miscellaneous implementation tasks . . . 33

5 Privacy-Preserving Cohort Exploration 35

5.1 MedCo Detailed Workflow . . . 36

5.2 Glowing Bear Query Construction . . . 36

5.2.1 Implementation . . . 37

5.3 IRCT Query Broadcast . . . 38

5.3.1 Implementation 38

5.4 MedCo Query Processing 39

5.5 Glowing Bear Result Processing 39

5.5.1 Implementation 40

6 System Evaluation 41

6.1 Requirements Fulfillment 41

6.2 Limitations 42

6.2.1 Combining Data Sources 42

6.2.2 tranSMART Support in the Interoperability Layer 42

6.2.3 Contributions to the Open-Source Community 43

6.2.4 Features Offered through PIC-SURE 43

7 Conclusion 44

7.1 Future Work 44

A Online Resources 49

A.1 Git Repositories 49

A.2 Other Resources 50

B Docker-Based Deployment 51

B.1 Docker Images 51

B.1.1 i2b2 51

B.1.2 i2b2 MedCo 52

B.1.3 IRCT 52

B.1.4 PostgreSQL Database Server 53

B.1.5 Lighttpd Web Server 53

B.2 Docker-Compose Run Configuration 53

Section 1

Introduction

To realize the promise of precision medicine, researchers need to freely explore large and diverse clinical and genomic data [21]. There exist several widely used technical solutions to explore those data sets, the two main open-source players being i2b2 [6] and tranSMART [14].

However those clinical research platforms are generally deployed at single or at a group of clinical sites, each hosting their own data. This leads to a situation where multiple fragmented systems exist at different locations all over the world. Since clinical sites have invested a lot of effort and money in those systems, this situation will not change anytime soon. Due to this scattered data, researchers find themselves limited by the amount of data that their own clinical sites produce.

Straightforwardly data sharing between clinical sites is a solution to this problem. There are two ways to approach data sharing, first is through centralization and second is through federation. Centralization is not considered because of the substantial efforts, thus cost, of doing so, in addition to the serious privacy concerns this implies.

By elimination, only the second approach, data sharing through federation, appears feasible. It however is challenging for two reasons. The first challenge is of technical nature; there is no interoperability between the different clinical research platforms. The second challenge is legal and ethical, as medical data are highly sensitive information about individuals, which are protected by strict regulations such as HIPAA [2] in the USA or GDPR [24] in the EU. Therefore providing strong privacy and security guarantees is absolutely crucial.

This thesis proposes a system that takes a step towards the technical convergence of those systems by providing an interoperability layer for i2b2 and tranSMART, exploitable by a unique front end, Glowing Bear [27]. In addition, this interoperability layer offers the ability to securely share medical data between clinical sites using the privacy-preserving system MedCo [23].

1.1 Objectives

We aim to provide a technical solution to enable researchers to do *cohort exploration*, i.e. explore patient medical data, from the clinical research platforms *i2b2* and *tranSMART* from a common front end. In order to expand further the pool of potential data accessible through this front end, we also aim to support *MedCo* which provides strong privacy and security guarantees. Those systems can be hosted at arbitrary physical locations. We do not seek to be able to combine patient data from different data sources, as it is a whole different class of problem. Which means we restrict ourselves to cohort exploration using one clinical research platform at a time. We do aim however to lay the technical ground to allow that in the future with further development, by building an extendable system.

For our front end, we target basic patient cohort exploration features, through queries resulting in a patient set size. Queries should support inclusion and exclusion criteria, themselves composed of basic presence of a term in the database, or using some categorical and numerical values.

An example of such a query supported by our system would be querying for *the number of patients aged between 17 and 30, diagnosed with cutaneous melanoma, excluding patients that had a chemotherapy* With the specific criteria supported by our system, this would translate to support the following query:

- *Inclusion criteria*
  - Age: ≥ 17, ≤ 30
  - Cutaneous Melanoma diagnosis
- *Exclusion criteria*
  - Chemotherapy

1.2 Solution Requirements

We translate the aforementioned objectives into the following requirements. Our solution must:

1. offer a modern web-based front end for clinical research platforms that allows cohort exploration based on inclusion and exclusion of query terms
2. be compatible with the two major open-source clinical research systems: tranSMART (v17.1) and i2b2
3. enable sharing of sensitive data in a privacy-preserving way with MedCo
4. be easily extensible for future support of additional platforms
5. alleviate technical hindrances against its use, by:
   (a) being easy to deploy, even in existing environments
   (b) not degrading the user experience in existing systems
   (c) enforcing secure authentication
   (d) being open-source
   (e) having a practical runtime

1.3 Outline

First we offer background information necessary to understand this thesis and review the related work in section 2. We then show the architecture of our solution and its designing process in section 3. In sections 4 and 5 we detail the design and implementation of the solution, broken down in two objectives: creating an interoperability layer, and building on it to integrate a privacy-preserving cohort explorer. In section 6 we show that our solution fulfills the aforementioned requirements, and we finally conclude and propose future work in section 7.

Section 2

Background & Related Work

2.1 Background

This section offers some important background information providing context to what is presented in the later sections.

2.1.1 i2b2

I2b2 is a clinical research platform used for cohort exploration. The server is written in Java and using Apache Axis2 [25] for its web services, it exposes an XML API to interact with it. Its source code is organized by *cells* in a *hive*. The *CRC* (Clinical Research Chart) cell is the data repository containing the patients data, the *PM* (Project Management) cell contains all information about users and metadata about the data in the CRC, the *ONT* (Ontology) contains the ontology with which the data is structured.

It stores patients data in this relational database encoded in a star schema. This schema has a central table *observation fact* that records any kind of observation about a patient. This table stores a unique set of references to its satellite dimension tables, each containing information about the *patient*, the *ontology concept* observed, the *encounter* that led to the observation and the *provider* that did it. The data is structured around standard ontologies such as *ICD-9* [3].

I2b2 has two official clients. First the *webclient* is a Javascript application running is web browsers and built around the *yui* [45] framework. Second is the *workbench*, a Java desktop application doing the same thing and a bit more. Both communicate with the server with the i2b2 XML API.

2.1.2 tranSMART

TranSMART is also a clinical research platform, offering cohort exploration but also more advanced analysis on these features. It is written mostly in Groovy [40], which runs in the Java Virtual Machine, and offers a modern JSON API in its latest version 17.1.

The core data schema is the same as i2b2 with two additional dimension tables: *studies* which represent a medical study from which data has been obtained, and *trial visits* recording progression through time of a study. Its data is structured around the *studies*, which each have their own ontologies. Some ontology concepts can be shared among studies and can be used as *cross-study concepts*.

Its official client is a web application rendered in Java on the server-side called *tranSMARTApp*.

2.1.3 PIC-SURE API

The PIC-SURE (Patient-centered Information Commons: Standardized Unification of Research Elements) [20] [38] [39] API is a generic API with a SQL-like syntax that aims to query different patients-centered data warehouses.

PIC-SURE is *resource*-based: each data source (e.g. i2b2) is considered a resource. To query these resources, the user of the API uses *clauses* submitted to the API *Query Service*. Four types of clauses are supported: *select*, *where*, *process* and *join*. Here we focus only on *where* clauses, which are the only ones used with i2b2.

The *where* clause is used to specified constraints on the queried data. The resource declares the *predicates* that its *where* clauses support. To give input to the predicates, the predicates declare *fields*. Each of those fields take a value as input, whose type is declared. Example of a *where clause*:

```
"where": [ {
    "field": {
        "pui": "/resource/study/Age/",
        "dataType": "INTEGER"
    },
    "predicate": "CONSTRAIN_VALUE",
    "fields": {
        "OPERATOR": "GT",
        "CONSTRAINT": "20"
    }
} ]
```

In order to construct queries, the resources expose a tree of query terms uniquely identified by a path, through the API *Resource Service*. Each of those query terms, if they are queryable, declare a data type. Each of the predicates used for the *where* clauses also declare one or more supported data types: this is the mechanism used to know which query terms support which predicates. In order to link the tree nodes together, the resources declare what kind of relationships between nodes exist. For example if the tree supports the relationship `CHILD`, a client can request all the children of a certain node.

IRCT

The official implementation of the PIC-SURE API is called Inter Resource Communication Tool (IRCT), and is the combination of four different components that are open-source and available at [35]. They are implemented in Java and use standard technologies: web application archive (WAR) [44] for deployment, Hibernate [31] for data storage. First the Communication Layer (IRCT-CL) implements the RESTful PIC-SURE API. The core component is the Application Programming Interface (IRCT-API), it handles the execution of queries and processing of results. An instance can be extended using the IRCT Extension (IRCT-EXT) that provides hooks and additional features without having to modify the core code. Finally the Resource Interface (IRCT-RI) connects to the different resources through connectors. Resources are registered and defined in the database. Their definition include all the predicates, operations, relationships, etc. they support.

2.1.4 Glowing Bear

Glowing Bear is a front end web application built using Angular as an new user interface to tranSMART 17.1. Angular is a Typescript [11] framework to develop web applications. Typescript is a language that when compiled produce javascript code to be run inside a web browser. Glowing Bear offer basic cohort exploration features, but also advanced analytic features or data export.

Its source code is organized in mainly Angular services and components. Services are singleton objects that handle logic related to data. Among others the notable services in Glowing Bear are the *Tree Node Service*, handling data relate to the tree of query terms or the *Resource Service*, handling communication with the back end. Components contain the logic of representing the data, and associate an object that contain the component logic, and a HTML template that displays it.

2.1.5 OpenID Connect

OpenID Connect is a protocol enabling authentication and authorization using a third-party implementing the server-side of the protocol. It offers several types of negotiations (called *flow*) between the server and client(s), but they all end up with an access token used by a user to access a resource. This token is standardized as a JSON Web Token (JWT) [17].

JSON Web Token

A JWT is three distinct base64-encoded values separated by a dot: two JSON objects and a signature. The first is the header, which contains metadata about the JWT, such as the signing algorithm used and the identifier of the key used to perform the signing. Example:

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "eTFrdyrNxXLNHI7pOYwybc7z1SBHTEcqWcMTybtdvQY"
}
```

The second is the payload of the JWT, and contains the identity of the authenticated user, its authorizations, and can contain any kind of custom fields set up by the administrator of the OIDC server. Other standard fields include expiration time, client identifier, token issuer, etc. Those data are called the JWT claims. Example:

```
{
  "jti": "6fd4f480-05ce-471f-a4ef-f35cb6a8e0d0",
  "exp": 1523454086,
  "nbf": 0,
  "iat": 1523453186,
  "iss": "http://localhost:8081/auth/realms/master",
  "aud": "glowing-bear",
  "sub": "df110f80-fa32-4174-970d-e42a7b24ae9f",
  "typ": "Bearer",
  "azp": "glowing-bear",
  "nonce": "N0.28573339803406971523453198656",
  "auth_time": 1523453186,
  "session_state": "74cd6393-9d24-4140-a87a-9e557f45499b",
  "acr": "1",
  "resource_access": {
    "account": {
      "roles": [
        "role1",
        "role2"
      ]
    }
  },
  "preferred_username": "test",
  "email": "test@test.com"
}
```

The last value is a signature generated by the OIDC server that serves two purposes: verify that the issuer was indeed the OIDC server, and protect the integrity of the token. Several algorithm can be used to generate this signature, here we use the algorithm *RS256*, i.e. the encryption of a *SHA256* hash using the asymmetric cryptosystem RSA.

2.1.6 MedCo

MedCo is a system of federated clinical sites sharing data together in a secure and privacy-preserving way. Its security relies on a collective authority [18] formed by all the MedCo nodes. It uses an homomorphic cryptosystem, which means that some operation on encrypted data is possible. Each of the node has a public and private key, and by assembling all the public keys together the public key of the collective authority is generated. The corresponding private does not exist, thus all operations requiring the private keys are distributed across all nodes using specific protocols.

For more detailed information about MedCo and its underlying technologies, please refer to [23].

2.2 Related Work

Our literature review focus on approaches similar to our main contribution, an interoperability layer for clinical research systems coupled with a front end.

SHRINE SHRINE (Shared Health Research Informatics NEtwork) [4] is a system that allows to network i2b2 instances together. From a single webclient, a fork of the i2b2 webclient, several i2b2 can be queried at the same time, and their results be displayed in a unique interface. By this, SHRINE offers an interoperability layer over an arbitrary number of i2b2 instances. SHRINE's design is very tightly linked to i2b2 and extending its support to additional clinical research system is not possible.

Borderline *Borderline* [22] is a set of services that includes a user interface, and several back end components. The user interface allows users to query clinical research platforms. This UI communicates with its back end which manages the queries and proxies them to the connectors. The connectors are responsible to query platforms containing patient data, such as tranSMART. Borderline is a more of a query workflow management tool rather than a proper cohort explorer, and it is at the time of writing still a work-in-progress.

Section 3

System Architecture

To design a system fulfilling the requirements listed in 1.2, we start from the bottom by first choosing its basic building blocks, and then we explore the different ways in which they can be assembled. Finally we show an overview of the final system. By the requirements, we only consider open-source technologies.

3.1 Choice of Open-Source Technologies

The choice of the open-source technologies to use depends mainly on the clinical research platforms that we aim to support. First we review those platforms, then we explore existing open-source front end systems in hope of determining if one of them fits the requirements of our solution. Finally we motivate the delegation of identity and access management to an identity provider, and choose a software that implements it.

3.1.1 Clinical Research Platforms: i2b2, tranSMART and MedCo

A clinical research platform is a back end software (i.e. running on a server and serving requests to clients), that stores any kind of medical data and can answer queries, with the purpose of identifying patient cohorts, and possibly performing analysis on those data.

Queries on this data can be for any purpose. Here follows two example use-cases for this kind of platform.

- An oncologist at a hospital tries to understand the response to specific medications according to criteria such as the presence of certain alleles in the genome of the patients.
- A pharmaceutical company wishes to conduct a study on a new medication, and needs to recruit a cohort of young patients that had a specific diagnosis at some point in time.

Our work fulfills the basic technical requirements of these two use-cases: obtaining patient counts based on inclusion and exclusion criteria. Other types of processing on the resulting patient groups and their data points, such as statistical analysis, are not the focus of this work.

The two main players in terms of open source clinical research platforms are *i2b2* [6] and *tranSMART* [14]. They enable the use-cases presented before by leveraging an almost identical database star schema, depicted figure 3.1. This schema has a central table containing *observations*, and several dimension tables: *patient*, *visit*, *concept* and *modifier*. And specifically for tranSMART: *study* and *trial visit*. An observation is a dated record of an event that refers to entries in the aforementioned dimensions. We choose to support these two platforms, in their latest version at the time of the project, as together they host a very large amount of data from hospitals, research institutes or pharmaceutical companies [32] [42].

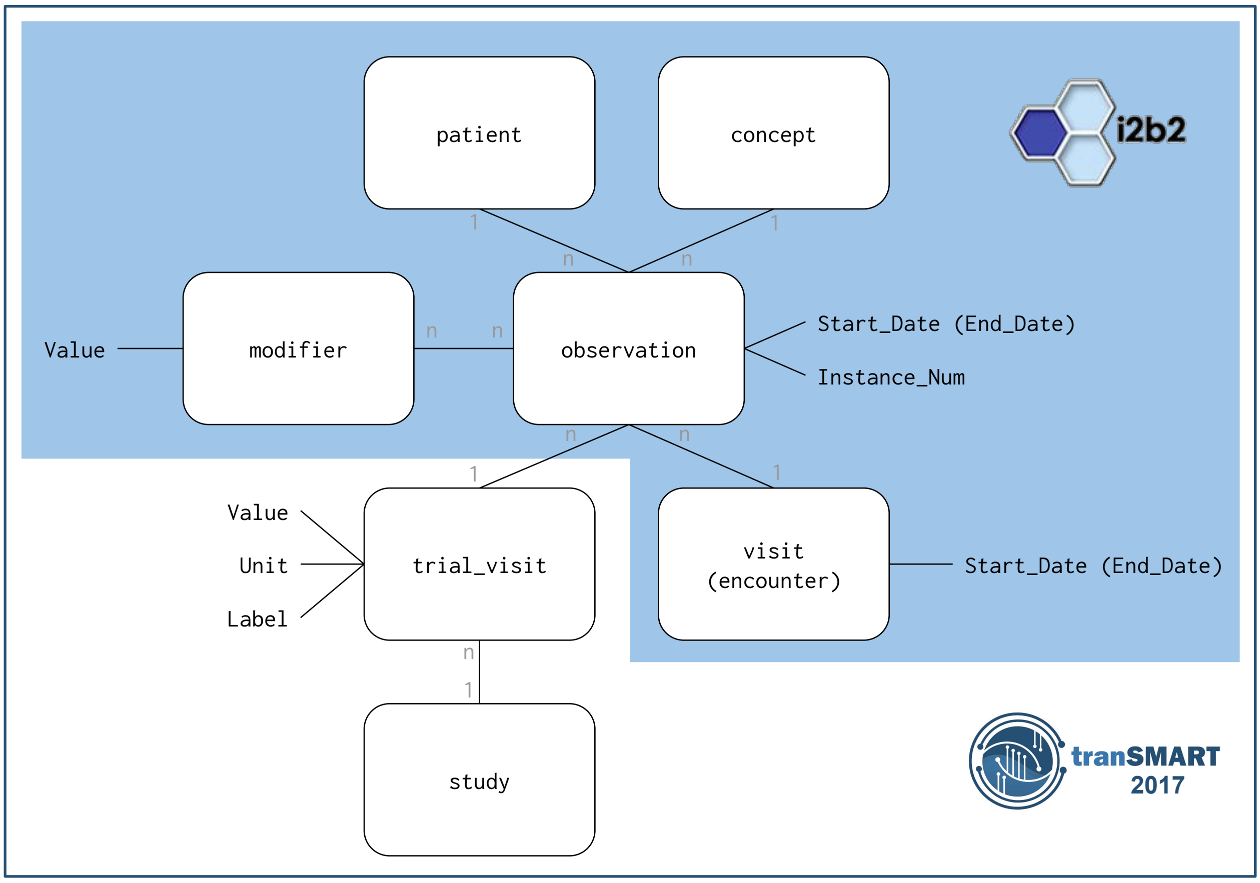


Figure 3.1: *i2b2* and *tranSMART* star schemas. Source: *Ward Weistra, The Hyve*

MedCo [23] is a privacy-preserving clinical research platform, and at the time of the project the only one that offers privacy-preserving guarantees. It enables the sharing of sensitive medical data within a federated group of clinical sites in a privacy-preserving way by using homomorphic encryption and obfuscation techniques. MedCo fits our requirements and use-cases as it is open-source, based on i2b2, and supports count queries based on inclusion and exclusion criterion, with end-to-end encryption.

3.1.2 Front End System: Cohort Explorers

To build our front end we have several systems on which we could base our solution on. We require it to be modern, open-source, web-based for portability, and if possible at least partially compatible with the clinical research platforms we support. If all those requirements can not be met, there is left the option of building the front-end from scratch, but as explained in this section we manage to avoid this significant effort.

The result of a review of available open-source front-end for cohort exploration who deserve to be investigated follows:

- i2b2 webclient [6] [33]
- i2b2 workbench [6] [34]
- transmartApp [14] [43]
- Glowing Bear [27] [28]

Each of the previously retained platforms have their official web front end: the *i2b2 webclient* for i2b2, and *tranSMARTApp* for tranSMART. Both of them are perfectly compatible with their respective back ends, however they suffer from two major problems: 1. they use old or outdated technologies, making future development and maintenance more difficult; 2. they are very tightly linked to their back end, making the development of support for the other difficult. The *i2b2 webclient* uses *yui* [45], a javascript framework developed by Yahoo, and not supported anymore since 2014. *tranSMARTApp* is a web application rendered on the server-side in Java, alongside tranSMART, and not a pure web client. For those reasons we choose to not retain those solutions. We can also consider the *i2b2 workbench*, client desktop application for i2b2, but as it is not web-based we discard it as well.

Glowing Bear is the front end system we choose to retain. It is modern, web-based and mature software. It is implemented on top of tranSMART (v17.1) but its source is structured generically, facilitating the implementation of support for other platforms. It thus fits all our requirements for a front end.

3.1.3 Identity Provider

Such a system with different components calls for a common and standardized way of handling authentication and authorization. The de-facto modern standard for this is the OpenID Connect protocol [13] (OIDC), which we are using in our solution. OIDC allows to externalize in a secure way authentication and authorization in a multi-components system. A stable, mature and open-source implementation of OIDC is the server software Keycloak [36], that we choose to use.

3.2 Building the System Architecture

3.2.1 Assembling the Building Blocks

Now that we have the basic building blocks of our solution, we design a system where they fit together in a coherent way. The architecture can roughly be categorized in three zones:

- Front end
- Interoperability layer
- Clinical research platforms

Our building blocks lie in the front end and clinical research platforms zones. We now are designing the interoperability layer that connects those two zones.

This interoperability layer can be implemented in the front end, in the back end or in both. Because the front end Glowing Bear implements the compatibility with tranSMART, one possible approach would be to implement the interoperability layer into Glowing Bear by adding the support for i2b2 and MedCo, using the i2b2 XML API, alongside tranSMART.

However, we want to avoid a fully front end i2b2 API implementation for the following two reasons: 1. the i2b2 XML API is old and cluttered, which means difficult to implement and maintain; 2. by only implementing the support for specific systems, this would not leave much space for extensibility.

The second option is implementing the layer fully in the back end, which would mean that:

- we need a back end component to take care of the APIs translation
- the support of tranSMART in Glowing Bear would be removed and re-implemented in the back end component

While the first consequence is not a problem, the second implies pretty heavy additional work, with not much added value.

For the reasons enumerated before, we choose in our system a hybrid approach: implementing the interoperability layer in both the front end and the back end. In the front end the support of tranSMART is kept, and we use a back end component to take care of the translation to the other clinical research platforms, i2b2 and MedCo, all the while having Glowing Bear communicating with this component with a unique API.

For this task the *PIC-SURE* API [39] is the perfect tool. It is a generic API that was designed to be translated to any kind of API used for querying medical data, and supports several systems, including i2b2. Although recent, it has been already successfully used to explore data [19]. The back end component implementing the PIC-SURE API is *IRCT* and is open-source. Thanks to its extensibility capabilities, we are also using it to implement support for MedCo. Support for the additional systems that IRCT is compatible with is left for future work.

To summarize, our interoperability layer is composed of

1. an interoperability module in Glowing Bear, which has support for both the tranSMART REST API v2, and PIC-SURE;
2. IRCT, which has support for i2b2 and MedCo.

3.2.2 Technical Choices Summary

We summarize our technical choices:

- Supported clinical research platforms: *i2b2 1.7*, *tranSMART 17.1*, *MedCo*
- Interoperability layer for i2b2 and MedCo: *IRCT* (PIC-SURE API)
- Front end with support for tranSMART REST API v2 and PIC-SURE: *Glowing Bear*

Technical Considerations

Additionally we take into consideration some software engineering good practices, which allow us to meet or alleviate the technical constraints listed in the requirements.

1. A practical query run time: the overhead of using our solution compared to the original clinical research platforms should be negligible.
2. Extensible: building support for additional systems should be facilitated.
3. Preserve existing user experience: the existing technologies that are used should not be modified in a way that degrades existing features.
4. Using standards: use existing standard APIs and technologies to the best extent possible.
5. Quality: produce quality code to facilitate maintenance and encourage reuse.

3.2.3 System Overview

Figure 3.2 shows the architecture of our solution as described in this section. The high-level idea of the workflow is the following:

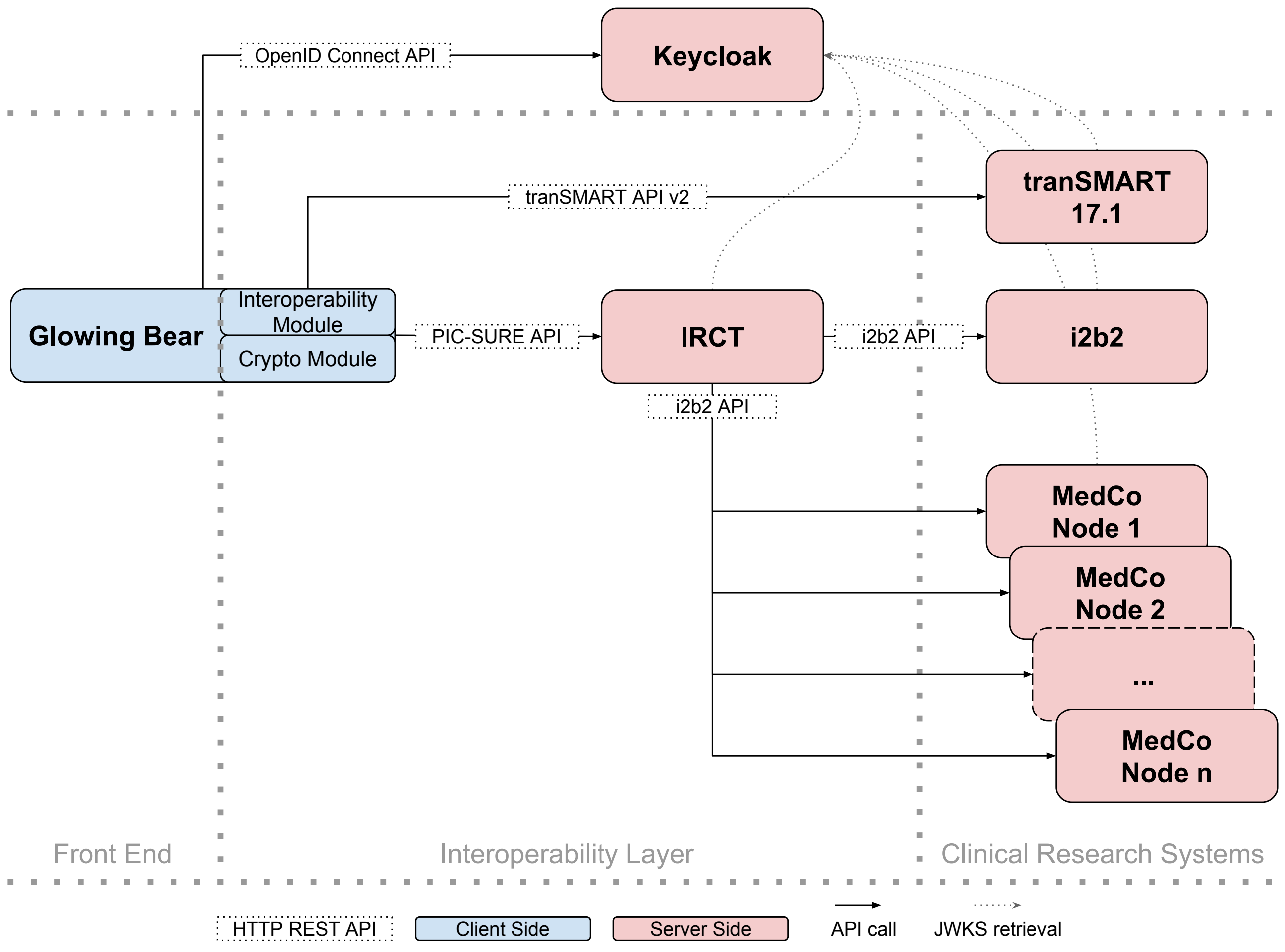


Figure 3.2: Full System Diagram

1. While loading Glowing Bear, the user logs in with Keycloak.
2. The user constructs a query, and Glowing Bear submits it through the appropriate channel:
   (a) *tranSMART*: from Glowing Bear, the query is immediately submitted to tranSMART.
   (b) *i2b2*: from Glowing Bear, the query is submitted to IRCT using the PIC-SURE API, and IRCT submits to i2b2 the query
   (c) *MedCo*: using the cryptographic module in Glowing Bear, the query is submitted to IRCT and then broadcasted to all of the MedCo nodes.
3. The result is fetched and displayed to the user in Glowing Bear. In the case of MedCo the cryptographic module is used to decrypt the result.

The design of step 1 is covered in section 4.2, step 2b in section 4.3 and 4.4, step 2c in section 5 and finally step 3 in section 4.3. Step 2a is not covered in this thesis as the support for tranSMART in Glowing Bear is preexisting.

Section 4

Interoperability Layer for Clinical Research Systems

Now that we have the architecture of our solution, this chapter covers the design and implementation of the first objective of our project: creating an interoperability layer for the clinical research systems i2b2 and tranSMART 17.1, that is exploitable by the front end Glowing Bear, and with the identity and access management handled by Keycloak. Support for MedCo is covered in the next section.

We are first going over the detailed workflow of the system, showing the end-to-end processing of a query. We are then detailing the design and implementation of the different parts solving the objective: the identity provider, the front-end, the API translation, and the processing in the back end systems. Appendix B covers exhaustively the technical details of the Docker-based deployment of the system.

4.1 Detailed Workflow

This section gives an overview of the full workflow of our system when constructing and executing a query. Some steps are specific to *tranSMART* 17.1 or *PIC-SURE*. The step numbering refers to figure 4.1.

1. **User Login**: The user accesses Glowing Bear through its web browser, which redirects it to the Keycloak login page. The user submits her credentials to Keycloak, which upon success redirects the user to Glowing Bear, giving at the same time the JSON Web Token (JWT) [17], containing authentication and authorization information. The user is thus logged in in the system.

2. **Glowing Bear Initialization**: Glowing Bear reads from its configuration which type of back end it should use. The tree of query terms is loaded, either entirely (*tranSMART*)

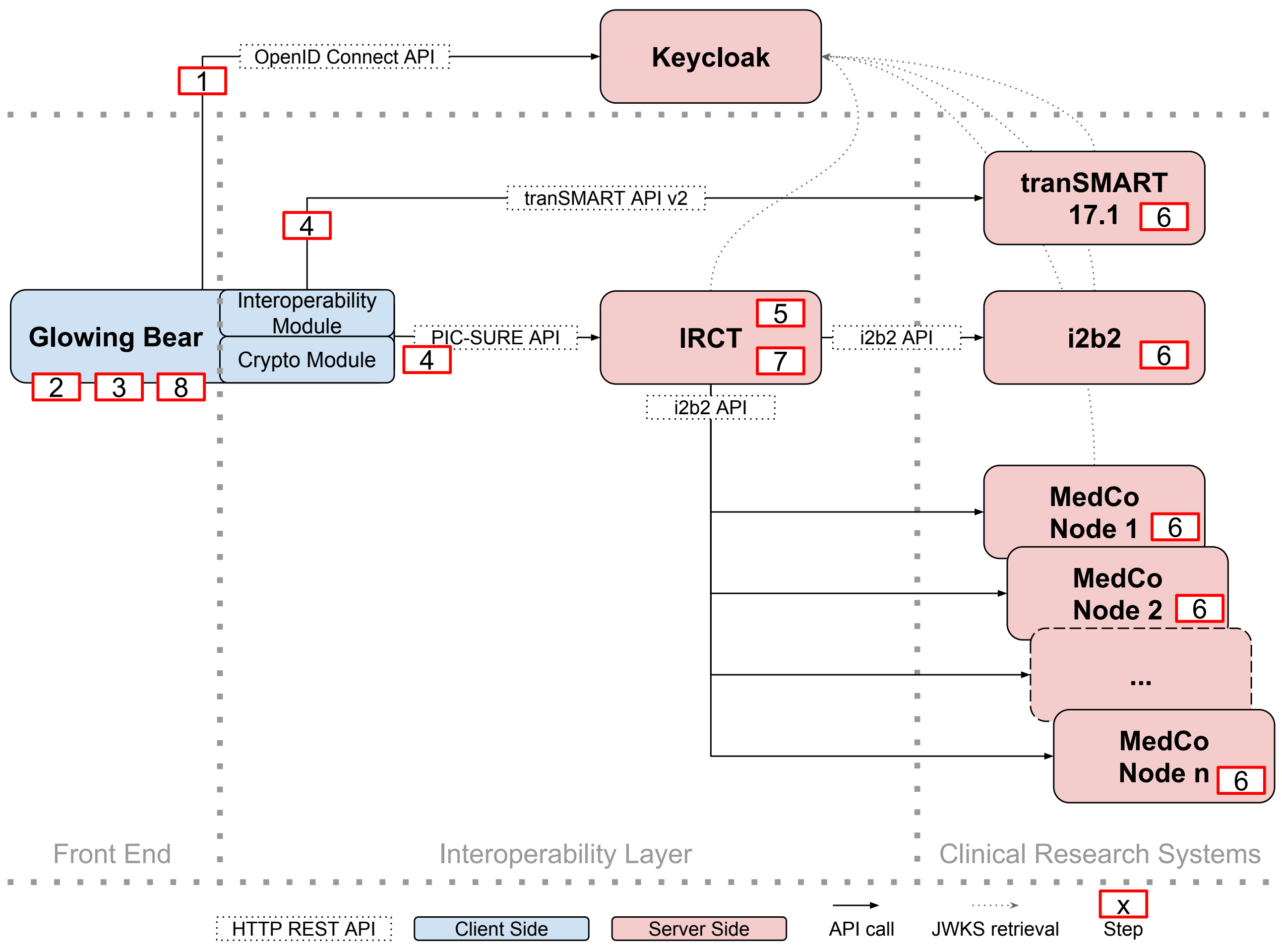


Figure 4.1: Full System Diagram, with execution steps

or only the root nodes (*PIC-SURE*). In the *PIC-SURE* case, Glowing Bear fetches the definitions of the data sources set up on IRCT.

3. **Query Construction**: The user browses the tree of query terms and uses them to construct a query corresponding to a patient set. When adding a term into the query panel, Glowing Bear might, according to its type, make a background request to fetch the term metadata, which is the case for example for categorical or numerical terms. The user then optionally sets value(s) to the query term.

4. **Query Submission**: Upon user request (or automatically according to the configuration), Glowing Bear submits the query to the back end.

5. **Query Translation** *PIC-SURE only*: IRCT uses the appropriate data source interface to perform the translation into the native i2b2 API of the query, and submits it to i2b2.

6. **Query Processing**: the submitted query, either directly from Glowing Bear for tranSMART or through IRCT for i2b2, is processed by the platform. The result is sent back to the requester.

7. **Result Storage** *PIC-SURE only*: IRCT stores the result in its database, and sets the query as being successful.

8. **Result Display**: the result is displayed to the user in Glowing Bear. Given the asynchronous nature of IRCT when processing queries, Glowing Bear periodically (every second) fetches the status of a *PIC-SURE* query, and only after fetches its result.

4.2 Identity and Access Management

This section first exposes how the authentication (identity) and authorization (access) are handled in our system with OpenID Connect and Keycloak, and then the steps taken to achieve this implementation.

4.2.1 Authentication

Authentication management is about verifying the identity of the user trying to access data. With OpenID Connect, the authentication of the user is made by the identity provider, here Keycloak. Keycloak can authenticate users in a number of ways: classic credentials (user and password) stored in its database, against external directories like LDAP, delegating to external OAuth2 or OIDC providers, and more. Two-factors authentication using One-Time Password (OTP) [8], time (TOTP) or counter (HOTP) based, is also available. This configuration, which can be specific to the client systems, is the responsibility of the administrator.

When loaded, Glowing Bear checks for the presence and validity of the JWT. If not valid the user is redirected to the login page of Keycloak. In this redirection several parameters are passed:

- the client identifier (`client_id`): identify for which client system the authentication is made for. Here the client systems are set to be the clinical research systems, as they are the ones that enforce control over who can access their data. For example if Glowing Bear is set to use an i2b2 instance through PIC-SURE, the client could be `i2b2-instance-1`.
- the redirection URL (`redirect_uri`): the URL to which Keycloak should redirect the user after login, here it is set to be the URL of Glowing Bear.
- the type of token requested (`response_type`): the type of token requested, this defines which flow is used (see below). We use `code` in order to use the *authorization code* flow.

Once the authentication is successful, several behaviors (called flows) can happen between the authentication of the user and the verification of the token at the client system. We focus on the behavior we use, the *authorization code* flow. Keycloak redirects the user back to Glowing Bear, giving the authorization code at the same time. Glowing Bear then exchanges this authorization code for the following tokens in the background:

- an *access token*: this is the signed JWT that contains the identity and authorizations of the user. It is sent along with the HTTP requests made to the back end systems. An example of a JWT is provided in section 2.1.5.
- a *refresh token*: when the access token expires, Glowing Bear uses the refresh token to request a new one. This can be done in the background without user interaction as long as the login session with Keycloak is still valid.

Glowing Bear embeds the access token into every HTTP request that needs to be authenticated. It is done by adding the following HTTP header, which is specified by [10]:

```
Authorization : Bearer <token>
```

Validating an access token in the JWT format requires several checks, performed in the back end systems [13]. The prerequisite to those checks is that the back end system should fetch from Keycloak its public signing keys, which are made available in the JSON Web Key Set format (JWKS) [15]. The checks are:

- the issuer of the token should be the expected Keycloak instance (`iss` field)
- the client identifier should be the identifier of the expected back end system (`aud` field)
- the token must not be expired (`exp` field)
- the signature of the token should be valid (`RS256` signature: SHA-256 with RSA signature, in JSON Web Signature (JWS) [16] format)

Using this authentication mechanism, the back end systems have a stateless authentication that needs only Keycloak's URL, the client identifier and the token.

4.2.2 Authorization

Authorization management is about restricting access to data depending on the access level of the user. With OIDC, the authorizations of a user are embedded as a *claim* into the JWT by the identity provider. While the authorizations are set at the identity provider, their enforcement are the responsibility of the client system, in this case tranSMART. Note that i2b2 and IRCT do not use the authorization capabilities of OIDC, only its authentication mechanism.

The standard does not specify a format for encoding authorizations, it is left at the client system's discretion. As Keycloak has its own format of authorization encoding, it also provides the ability to map authorizations to a specific format according to the client. It is thus the responsibility of the administrator doing the deployment to make these two formats match by using the mapping capabilities of Keycloak.

4.2.3 Implementation

4.2.3.1 Glowing Bear

Glowing Bear originally makes use of the client side of the OAuth2 [9] protocol for authorization with tranSMART. We thus modify it to use the OpenID Connect protocol. To embed the JWT into the HTTP requests an interceptor is implemented. It intercepts all the requests made by Glowing Bear, examines them, and adds in the headers the token if it is needed. This way, the requests can be authenticated by the receiving back end.

4.2.3.2 IRCT

In its original implementation IRCT supports only JWT with a `HS256` signature, i.e. a shared-secret based signature. Because Keycloak does not support this type of signature, and because the standard recommends to use `RS256` signature [13], we modify the IRCT implementation to do so. The support of `RS256` signatures is added alongside `HS256`, which instead of the shared secret needs as input the public signing keys of Keycloak. The retrieval of those in the JWKS format is thus implemented, from a URL specified in the configuration. The verification of the token is voluntarily incomplete: checking for the issuer and the client identifier is left as the responsibility of the back end system queried by IRCT.

4.2.3.3 i2b2

i2b2 initially supports authentication with its own user management mechanism or through LDAP or NTLM directories. We take advantage of the i2b2 Project Management (PM) cell ability to specify in the database the authentication mechanism to implement the support of OpenID Connect. This allows users authenticated with OIDC to cohabit with users authenticated with the traditional i2b2 way. This is done through the i2b2 PM parameters (as specified section 2.1.1), which can be applied either at user, cell or project level. They are the following:

- token issuer (Keycloak instance URL)
- signing public keys URL (JWKS)
- client identifier
- field name in the JWT of the user's username

This last field is needed to ensure that the user account in the i2b2 database matches the user specified in the token, in order to enforce the authorizations. Because there is no such thing as a standard username field, it needs to be specified by the configuration.

The token is passed through the `password` field of the i2b2 XML API. This approach is not among the recommended ones, but it does not go against the OpenID Connect standard, and it simplifies greatly the implementation.

4.3 Front End: Glowing Bear

The front end we chose to use, Glowing Bear, is modified to support the PIC-SURE API in addition to the existing support of the tranSMART REST API v2. This section describes the execution flow of Glowing Bear with PIC-SURE only, as the support for tranSMART is preexisting. For each part, we first describe the workflow in its final form, and then which implementation steps are taken to reach that stage.

4.3.1 Initialization

The very first step is reading the configuration, which defines the mode in which the running instance is running: tranSMART or PIC-SURE; and the corresponding authentication process to use. It also allows to enable or disable certain features, according to the back end compatibility. The features that can be controlled this way are: 1. query saving, 2. data table, 3. query subscription, 4. data analysis, 5. data export, 6. observation count, 7. variable selection, which are all disabled when using PIC-SURE.

After the configuration is loaded, Glowing Bear checks the validity of the token and redirects the user to the Keycloak login page if invalid or not present. After the login is successful Glowing Bear gets the list of available data sources with the PIC-SURE API and extracts the definition of the data source it is configured to use (see 2.1.3).

4.3.1.1 Implementation

The following configuration options are added:

- `endpoint-mode`: use tranSMART or PIC-SURE mode
- `picsure-data-source-name`: PIC-SURE data source to use
- `force-i2b2-nesting-style`: force the i2b2 AND/OR query format
- `enable-greedy-tree-loading`: controls whether the tree of query terms should be loaded entirely during initialization

- `authentication-service-type`: controls the type of authentication service used
- `oidc-server-url`: URL of the OpenID Connect server

- `oidc-client-id`: client identifier to use with the OIDC server

- `show-observation-counts`: controls the display of the observation counts
- `include-query-saving`: controls the use of the query saving feature
- `include-data-table`: controls the use of the data table feature
- `include-query-subscription`: controls the use of the query subscription feature
- `include-variable-selection`: controls the use of the variable selection feature
- `enable-analysis`: controls the display of the analysis tab
- `enable-export`: controls the display of the export tab

We make the code agnostic to the differences between tranSMART and PIC-SURE except from a single service, the *Resource Service*, which acts as an interface between the core Glowing Bear code and the API-specific code. The use of one or another service is controlled by a configuration option. We factor away into the *API Endpoint Service* the methods that handle the HTTP calls from the *tranSMART Resource Service*, so that they can be used in the newly implemented *PIC-SURE Resource Service.* At initialization this service fetches the definition of the PIC-SURE data sources available, and implements the PIC-SURE API calls.

4.3.2 Query Terms Tree

Right after Glowing Bear is fully initialized the tree of query terms is loaded. As PIC-SURE supports only node-per-node loading, just the root nodes are loaded. This behavior is controlled by a configuration option. For this reason, the free-text search in the tree can not be used and the auto-completion feature in the query construction component is degraded and only has the pre-loaded nodes. Then, as the user expands the nodes in the tree, calls are made to the back end to dynamically load the children nodes.

Each node in the tree is characterized by a unique path, which for PIC-SURE on i2b2 has the following format:

```
/<data source name>/<i2b2 project name>/<category>/.../<concept>/
```

When displaying the tree, we omit the first element of the tree (*PIC-SURE data source name*) as it is constant during the runtime, specified by the configuration.

Each tree node contains the information needed to be used as a query term. They can be of three types: *concept*, *modifier*, *study*, or *unknown*. An unknown type node is not queryable. A study type node is only used by tranSMART. A modifier type node is applied as a constraint to a concept. The queryable nodes in i2b2 are all concepts, and are themselves subdivided in several types:

- *Simple*: simple concept with no associated value

- *Categorical*: concept with a categorical (i.e. enumerated) value
- *Numerical*: concept with a numerical value
- *Text*: concept with a free-text value

Each of those types is mapped to the appropriate user interface component, for taking the user input. This component is displayed in the query construction component, after the tree node is drag-and-dropped from the tree. Additionally the nodes have the information about whether they are internal nodes or tree leaves.

4.3.2.1 Implementation

Originally Glowing Bear supports only a greedy loading method, that is the whole tree is loaded at initialization. Because PIC-SURE does not support this, at least not with acceptable performance, we refactor the *Tree Node Service* to load at initialization time only the root node, if specified in the configuration. Then we add in the *Tree Node Component*, which displays the tree in the UI, the ability to load dynamically the children nodes when the user expands a tree node with missing children.

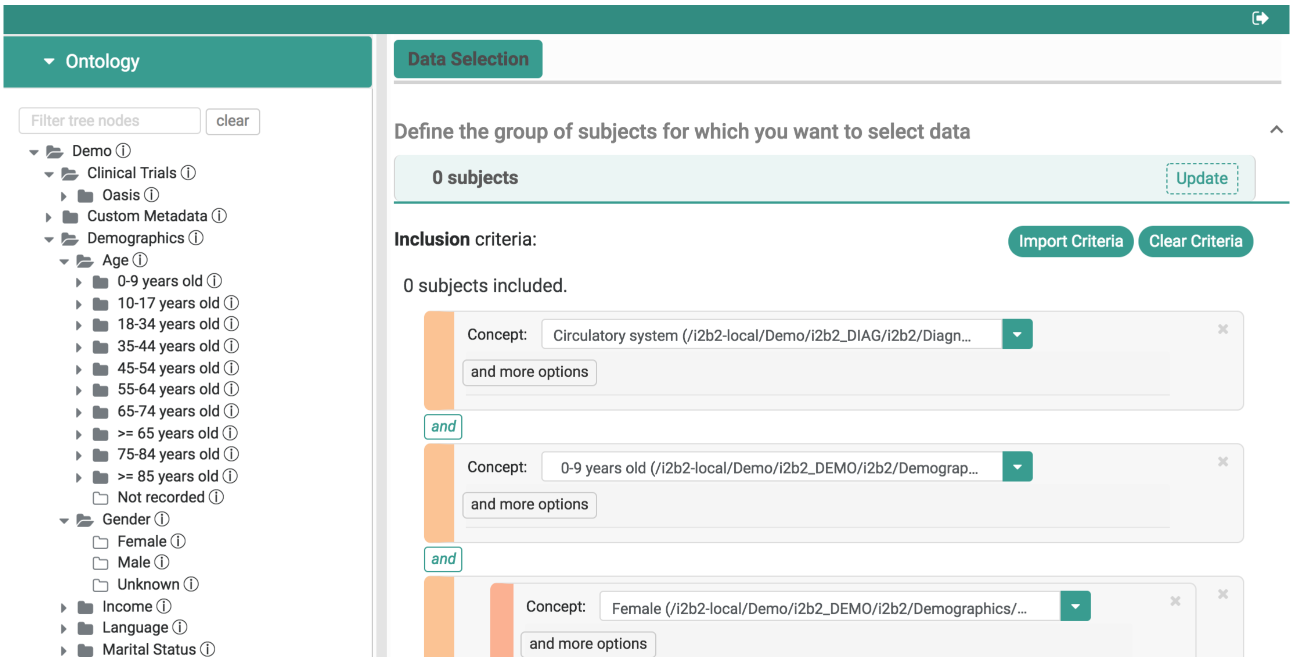
The original tranSMART tree node implementation represents tree nodes without typing, i.e. generic objects. To improve the code quality and enforce a common structure between the two API implementations, we create a new *Tree Node Model*. We modify the tranSMART and create the PIC-SURE API implementation so that they both produce the same *Tree Node Model*. Among the concept types originally supported, we add the types *categorical option*, *text*, *high dimensional* and *simple*. These cover both the new i2b2 types, and a regrouping of tranSMART types for more clarity.

4.3.3 Query Construction

The user constructs her query by drag-and-dropping query terms from the tree to the query construction component. A screenshot of Glowing Bear's UI when constructing a query is shown figure 4.2. As the user does so, Glowing Bear might request from the back end additional information on the query terms. Once done, and before submitting to the back end, this query is mapped to the PIC-SURE query format.

4.3.3.1 Query Terms Metadata

When a query term is dropped into the query construction component, a dedicated sub-component is created. This sub-component when initialized will, according to the query term type, make a request to the back end using a tree node call with the `AGGREGATE` relationship, on the term that is about to be used in the query. With an i2b2 data source, this call is used for *categorical* query terms types, as i2b2 does not have support to return aggregate values of numerical or date terms.


Ontology

Filter tree nodes clear

- Demo ⓘ
  - Clinical Trials ⓘ
    - Oasis ⓘ
  - Custom Metadata ⓘ
  - Demographics ⓘ
    - Age ⓘ
      - 0-9 years old ⓘ
      - 10-17 years old ⓘ
      - 18-34 years old ⓘ
      - 35-44 years old ⓘ
      - 45-54 years old ⓘ
      - 55-64 years old ⓘ
      - 65-74 years old ⓘ
      - >= 65 years old ⓘ
      - 75-84 years old ⓘ
      - >= 85 years old ⓘ
      - Not recorded ⓘ
    - Gender ⓘ
      - Female ⓘ
      - Male ⓘ
      - Unknown ⓘ
    - Income ⓘ
    - Language ⓘ
    - Marital Status ⓘ

Data Selection

Define the group of subjects for which you want to select data

0 subjects Update

Inclusion criteria: Import Criteria Clear Criteria

0 subjects included.

Concept: Circulatory system (/i2b2-local/Demo/i2b2_DIAG/i2b2/Diagn...

and more options

and

Concept: 0-9 years old (/i2b2-local/Demo/i2b2_DEMO/i2b2/Demograp...

and more options

and

Concept: Female (/i2b2-local/Demo/i2b2_DEMO/i2b2/Demographics/...

and more options

Figure 4.2: Query terms tree and query construction in Glowing Bear on i2b2

The answer is a usual PIC-SURE tree node answer, with additional metadata containing the categorical values. These are then used by the sub-component to be displayed to the user. This is an example of such an answer:

```
{
    "pui": "/<concept path>/",
    ...
    "attributes": {
        ...
        "aggregate.categorical.0": "<categorical value 0>",
        "aggregate.categorical.1": "<categorical value 1>",
        ...
}
```

4.3.3.2 Generation of PIC-SURE query

Data Types A query is a set of constraints, expressed as a list of *where clauses* as explained in 2.1.3. Since each tree node is associated to a PIC-SURE data type like explained in section 4.3.2, we know through the data source definition what predicates can be applied on them. For an i2b2 data source, here follows the data types and corresponding predicates:

- *Simple* concept: `CONTAINS`, `CONSTRAIN_DATE` and `MODIFIER` predicates
- *Categorical*, *Numerical* and *Text* concept: the above and `CONSTRAIN_VALUE`

Predicates Some predicates take values as input:

- `CONSTRAIN_VALUE`:
  - `OPERATOR`: the operator to apply on the value
  - `CONSTRAINT`: the value of the constraint
  - `UNIT_OF_MEASURE`: optionally the measure unit of the value
- `CONSTRAIN_DATE`:
  - `FROM_INCLUSIVE`, `TO_INCLUSIVE`: if the to / from date is inclusive
  - `FROM_DATE`, `TO_DATE`: date constraint boundaries
  - `FROM_TIME`, `TO_TIME`: if the to / from constraint should be applied on the start or end date of the observation
- `CONSTRAIN_MODIFIER`:
  - `MODIFIER_KEY`: key of the modifier to be applied on the concept

Example Assembling those together, a *where clause* based on a value would be similar to the following:

```
[{
    "field": {
        "pui": "/i2b2/project/laboratory/biochemistry/Creatinine (mg per dL)/",
        "dataType": "CONCEPT_NUMERIC"
    },
    "predicate": "CONSTRAIN_VALUE",
    "fields": {
        "OPERATOR": "GT",
        "CONSTRAINT": "4,2"
    }
}]
```

Combination of query terms While tranSMART supports full freedom in the combination of query terms, i2b2 is restricted to the following format:

```
    (A OR B OR ...) AND (C OR D OR ...) AND ...
```

That is, blocks of query terms linked by an `AND`, where each block is internally linked by an `OR`. The restriction to this format is controlled by the configuration, and if enabled the user is forced to respect it. In the PIC-SURE API, this translates to setting accordingly the *logical operator* as such:

```
[
    {
        <where clause definition>,
        "logicalOperator": "AND"
    }, {
        <where clause definition>,
        "logicalOperator": "OR"
    },
    ...
]
```

4.3.3.3 Implementation

The mechanism to request metadata about a query term from the back end is preexisting. As such we implement the call in the *PIC-SURE Resource Service* that returns the specific model that Glowing Bear expects to populate its data structures for the display and selection of the categorical variable.

The whole process of generating queries in the PIC-SURE format are implemented in a new utility class *PIC-SURE Constraint Serializer*. Models representing the *where clauses* are implemented, and the serializer is responsible to produce those, based on the Glowing Bear internal models that represent the query.

In order to restrict the format of the queries to the i2b2 format, a configuration option is added. If enabled, the component holding the combination of query terms enforce this restriction. The first level of nesting is always an `AND`, the second level always an `OR`, and no more levels are allowed.

4.3.4 Query Request & Result Retrieval

After constructing the query as previously described, Glowing Bear submits the query to the back end using the *PIC-SURE Query Service*. At this stage, we are interested only in the result patient count of the query to be displayed to the user, which is what is requested in the query.

4.3.4.1 Query Request

When using only inclusion criterion, a single query is submitted as expected. The count returned is displayed as it is to the user. However when using both inclusion and exclusion criterion, two query requests are actually submitted:

- Query 1: `<inclusion criterion>`
- Query 2: `<inclusion criterion> AND <exclusion criterion>`

Note that in the second query, the exclusion criterion is not negated. Making two separate queries allows to display slightly more detailed counts to the user:

- Inclusion count: `<Q1 count>`
- Exclusion count: `<Q2 count>`
- Query total count: `<Q1 count> - <Q2 count>`

4.3.4.2 Result Retrieval

Due to the asynchronous nature of the PIC-SURE queries, the request does not include the actual result, but the identifier of the result. Using this identifier, Glowing Bear regularly polls the back end using the *PIC-SURE Result Service* to inquire about the result status. This status can take several values:

- `RUNNING`: the query is still running, the regular polling continues
- `AVAILABLE`: the query has successfully finished and the result can be requested
- `ERROR`: the query is in error state, the polling stops and the user is informed of the error message that comes along

When the result becomes available, it is requested using the same identifier and then displayed.

4.3.4.3 Implementation

The implementation of this part is made in the Glowing Bear *PIC-SURE Resource Service*, orchestrated from a single method as the core code expects synchronous count results from the query. The result retrieval is configured to poll every second the back end for the result status, and waits up to one minute. If after one minute the result is not available, it is considered as failed.

Additionally, because i2b2 supports only patients count and no observations count, the display of this value is disabled through the configuration.

4.4 PIC-SURE to i2b2 API Translation

On the back end side, to provide an interoperability layer with i2b2 and later more systems, we use IRCT [35] (Inter-Resource Communication Tool). IRCT implements the PIC-SURE API, which is used by Glowing Bear. Besides that it can support a multitude of systems, but we focus here on the support of i2b2.

This section covers the process of API translation happening in IRCT, from the PIC-SURE API to the i2b2 API. We go over the full query execution flow: browsing through query terms, executing a query and getting back its results.

4.4.1 Query Terms Tree

4.4.1.1 Tree Structure and Nodes Relationships

Query terms are exposed through a tree to client systems. Each node in the tree is uniquely identified by a path, with at its root the different data sources available in the instance. When a client requests the children of the root path `/`, the nodes returned correspond to the configured data sources in the instance. Below this root level, the tree is data source implementation-dependent. We review here the tree of an i2b2 data source.

The data sources tree structure is based on *relationships* between nodes. The most obvious relationship is the parent / child / sibling: based on a tree node, a client can requests its *children*, *parent* or *sibling* nodes. This allows the client system to browse the tree in an iterative fashion, which is what happens with i2b2. i2b2 itself already exposes a tree, so the mapping of the tree is one-to-one below a certain level:

```
/<data source>/<i2b2 project>/<i2b2 root node (category)/<i2b2 tree>/
```

Note that it is not possible to use query terms coming from different data sources at same time, or in the case of i2b2 from different projects.

On top of the previously mentioned standard relationships, a data source implementation can declare additional ones, which can be seen as adding a dimension to the tree. We exploit here this possibility in order to expose aggregate data about the nodes. Here we add an *aggregate* relationship to the *categorical* nodes, which is exposed to the client system. As such, the client system knows about this possibility, and can request the aggregate values of a node, in this specific case the values this categorical query term can take.

The relationships exposed by an i2b2 data source tree are:

- *Child*: returns the children node
- *Modifier*: returns the modifiers associated with the requested tree node

- *Aggregate*: returns the aggregate data of the tree node (such as the values of a categorical query term)

4.4.1.2 Tree Nodes as Query Terms

To determine how tree nodes can be used as query terms, we do the following. Each node of the tree can declare a type. If the node does not have a type, it is not queryable and is a simple ”container” node. When the node has a type it can be used as a query term with some predicates, and each predicate declares in the data source implementation which node types it supports. So we need to 1. translate correctly the i2b2 node types into PIC-SURE node types 2. declare predicates that support those types

The types can be defined by the data source implementation, or be one of the primitive types offered by IRCT. With the i2b2 implementation, we do not use the primitive types, only the ones we define:

- `concept`: simple query term, maps to the *simple* GB types
- `concept_numerical`: query term with numerical value, maps to *numerical* GB type
- `concept_enum`: query term with categorical value, maps to *categorical* GB type
- `concept_string`: query term with free-text value, maps to *text* GB type

4.4.1.3 Implementation

Browsing the i2b2 ontology tree is already implemented in IRCT, we enhance it with some features that we need. The specific i2b2 data types listed before are implemented, and the extraction of those types from the i2b2 ontology tree as well. We also implement adding to the tree nodes the relationships *child* if the node has children, and *aggregate* if it has aggregate values available. The i2b2 data source definition is modified accordingly.

Support for the *aggregate* is implemented by making an ontology request to i2b2, and parsing the XML metadata associated with the node that contain this information, to embed the categorical values in the tree node.

4.4.2 Query Translation

Here we show the process of translating a query from the PIC-SURE API format to the i2b2 API format.

4.4.2.1 Query Structure Translation

Recall that all the query terms are represented in the PIC-SURE API by a *where clause*. In the i2b2 terminology, a query term is an *item*, and a group of query terms linked by an `OR` is a *panel*. The *panels* are linked together with an `AND`. The translation process from the PIC-SURE to the i2b2 API iterates over all the *where clauses*, translates them into an i2b2 *item*, and assembles those *items* appropriately into *panels* according to the PIC-SURE logical operator used.

See below an example of a PIC-SURE query, with query terms `A`, `B`, `C` and `D`:

```
[
    {
        <A>
    }, {
        <B>,
        "logicalOperator": "OR"
    }, {
        <C>,
        "logicalOperator": "AND"
    }, {
        <D>,
        "logicalOperator": "OR"
    }
]
```

This query can be abstracted as seen below. Here a query term denoted by `*` is an *item*, and a block of query terms denoted by `---` is a *panel*.

```
     *    *       *    *
    (A OR B) AND (C OR D)
    --------     --------
```

This query would be translated into the following i2b2 XML query format:

```
    <panel>
        <item>A</item>
        <item>B</item>
    </panel>
    <panel>
        <item>C</item>
        <item>D</item>
    </panel>
```

4.4.2.2 Query Terms Translation

When translating a query term into and i2b2 *item*, some additional processing might be needed depending on the predicate used. It is not the case for the simple `CONTAINS` predicate, which is straightforwardly translated using the path in the i2b2 ontology tree. For the others, the additional information is added to the item:

- `CONSTRAIN_MODIFIER`: the path of the applied modifier in the ontology tree
- `CONSTRAIN_DATE`: the dates and parameters specified by the querier
- `CONSTRAIN_VALUE`: the value and parameters specified by the querier

4.4.2.3 Implementation

The translation of queries is left mostly unmodified from its original implementation, the only addition is the implementation of support for query terms negation.

4.4.3 Results Management

After making the query request, IRCT registers in its own database the query with an identifier that encodes the different corresponding identifiers on the i2b2 side:

```
<i2b2 project id>|<i2b2 query id>|<i2b2 result id>
```

Using those i2b2 identifiers IRCT is able to request from i2b2 the status of the query. Once the query is over and successful, its patient count results are retrieved from i2b2 and stored in the IRCT database.

4.4.4 Miscellaneous implementation tasks

While IRCT uses Hibernate [31] to abstract away the SQL database used, in our case using PostgreSQL does not work as the default SQL table names use some reserved keywords. To counter this we use the Hibernate feature that allows to add a specific prefix to the table names.

Configuring an i2b2 data source to be used in IRCT requires to add a specific set of data in the PostgreSQL database. To do so, we implement a PL/pgSQL function that can be called with the appropriate parameters to add an i2b2 data source.

Web browsers block by default web requests made to other web sites than the originating web site, as a security feature. This can be circumvented by using the Cross-Origin Resource

Sharing (CORS) [7] standard that modern web browsers implement. CORS allows web pages to make web requests to web sites other than the web site from which the script is executed, but only if the web service serving the request explicitly allows the web site from which the request originates. This is checked automatically by the browser, which before making an actual HTTP request, will make an `OPTION` HTTP request to the web service to get the list of domains allowed to make requests. This feature is not initially implemented by IRCT. We implement it by allowing unauthenticated HTTP `OPTION` requests, and by embedding the appropriate headers into the answer.

Section 5

Privacy-Preserving Cohort Exploration

At that point we have all the basis of our solution, including the interoperability layer. We are building on this to integrate the privacy-preserving cohort explorer MedCo in this section, in a way similar to i2b2. As MedCo is detailed exhaustively in [23] we will focus on its integration with our system, and will only summarize the major steps of execution. This section assumes knowledge of background information from section 2.1.6.

In the following section, after reviewing some necessary definitions, we give an overview of the MedCo workflow. Then we see the process of constructing a query in Glowing Bear for MedCo, how it is broadcasted to the MedCo nodes, the query processing in each of the nodes, and finally the handling of the result.

Definitions

We use in this section the following definitions:

Pk_u, pk_u	public and private keys of user u
Pk_i, pk_i	public and private keys of MedCo node i
Pk_C	collective authority public key
S_i	distributed deterministic tagging secret of MedCo node i
$\text{ENC}_k[a]$	homomorphic encryption of a with key k
$\text{DDT}_k[a]$	distributed deterministic tag of a with secret k
q_v	integer representing the query term v
R_i	result of MedCo node i
$f_j \in \{0, 1\}$	dummy flag of patient j

5.1 MedCo Detailed Workflow

We show here the detailed workflow of our system when using MedCo, which follows the structure of the i2b2 workflow (section 4.1) with some differences.

- **User Login**: *same as for i2b2.*
- **Glowing Bear Initialization**: *same as for i2b2.*
- **Query Construction**: The user browses the tree of query terms and uses them to construct a query corresponding to a patient set. For each query term, an integer q_v representing the term is extracted from its metadata and encrypted using the public key of the collective authority Pk_C, denoted $\text{ENC}_{Pk_C}[q_v]$.
- **Query Submission**: The query with encrypted terms $\text{ENC}_{Pk_C}[q_v]$ is submitted to IRCT through the PIC-SURE API.
- **Query Translation**: The query is translated the same way as i2b2, and submitted to all of the MedCo nodes at the same time.
- **Query Processing**: MedCo processes the query by computing the encrypted results encrypted with the user's key $\text{ENC}_{Pk_u}[R_i]$. Details of this step are in section 5.4.
- **Result Storage**: *same as for i2b2.*
- **Result Display**: Glowing Bear receives the encrypted results $\text{ENC}_{Pk_u}[R_i]$, decrypts them using the user's private key pk_u and displays them to the user.

5.2 Glowing Bear Query Construction

In this section we show how Glowing Bear constructs a MedCo query to be sent to IRCT through the PIC-SURE API.

The tree of query terms exposed through PIC-SURE is the same as it would be for i2b2: an i2b2 ontology cell is used to hold the MedCo ontology. The differences lie in the additional constraints and metadata added on the ontology at loading time, which we use in Glowing Bear to construct the query.

While browsing the tree from Glowing Bear, the IRCT data source implementation for MedCo sets the tree node types properly, according to the presence of a specific tag in the i2b2 ontology, flagging the presence of encrypted query term. The identifier of the ontology element contains this tag and the integer q_v:

```
ENC_ID:<integer representing the query term>
```

The MedCo tree node types are the same as the i2b2 types, with the difference that they indicate the encryption:

- `enc_concept`
- `enc_concept_numerical`
- `enc_concept_enum`
- `enc_concept_string`

Predicates exposed through PIC-SURE are the same as i2b2, except with the support of the additional MedCo data types. To use those predicates with encrypted query terms, we use a format that is recognized by the MedCo data source implementation in IRCT. To do so we embed in the query term path the base64-encoded encrypted integer the following way:

```
/<data source>/<MedCo project>/ENCRYPTED_KEY/<encrypted integer>/
```

5.2.1 Implementation

Implementation of the i2b2 tree browsing in IRCT is the same as the one used for i2b2, except for the data types that are modified on the fly. The modification is made by a simple mapper that maps the i2b2 data types to the MedCo ones, as from the i2b2 point-of-view those are the same. We implement the new data types for MedCo, which are the same as the i2b2 ones except that they are prefixed by `enc_` to represent the encryption of those query terms by MedCo.

To support the cryptographic primitives of MedCo in Glowing Bear, we use GopherJS [29] to transpile original MedCo code written in Go [41] to Javascript. The resulting Javascript library is packaged as a npm module [37] to be added to Glowing Bear dependencies. It supports generating a pair of public and private keys, encrypting and decrypting values. The support of those operations is added to the *PIC-SURE Resource Service* in Glowing Bear.

In Glowing Bear we implement two hooks that:

1. detects and encrypts query terms that should be encrypted,
2. detects and decrypts encrypted results.

To support the encryption with the collective authority public key Pk_C we add a configuration option that specifies the URL at which it should be retrieved. To support decryption with the user private key pk_u we add at the initialization of Glowing Bear the generation of a random pair of keys.

5.3 IRCT Query Broadcast

After Glowing Bear has submitted the query to IRCT through the PIC-SURE API, IRCT invokes the MedCo data source implementation to run the query with the MedCo nodes. Running the query with the MedCo nodes uses the i2b2 XML API, but does not reuse the i2b2 data source implementation like with the tree browsing, due to fundamental differences in how the query is executed.

The process to generate the query in the i2b2 XML API format is the same as in i2b2, described in section 4.4.2, even though the paths of the queried items are encrypted values. However the submission of the query to i2b2 is different, as the query has to be submitted to all the MedCo nodes simultaneously. This is needed as the MedCo nodes later need to synchronize between them to perform some collective cryptographic protocols. Note that the authentication is the same as for i2b2: the token is sent in the query, assuming that the requesting user is registered in all the nodes.

5.3.1 Implementation

We implement the part of the PIC-SURE data source implementation for MedCo that handles the querying of the data source. To implement the simultaneous querying of the MedCo nodes, we use as many threads as there are nodes and execute them at the same time. We use a *count down latch* to synchronize the threads and wait for their completion. After a set amount of time without answer from the MedCo nodes, they timeout.

The query is submitted directly to the MedCo nodes. This is different from the original MedCo behavior that relies on SHRINE [4]. In this previous implementation, the query was submitted to a single SHRINE node, and only then the query was broadcasted to the MedCo nodes. Here this broadcaster role is taken by the PIC-SURE data source implementation of MedCo. The reasoning behind bypassing SHRINE is that it is a multi-components software, that is complicated to set up and deploy, and expensive to maintain. Moreover it was not bringing a significant added value to MedCo.

Doing so, we lose a feature offered by SHRINE, which is the ontology translation that SHRINE operates between a common network ontology and a local i2b2 ontology. This is not a big problem, as anyway the mapping between the network and common ontology needed to be made and maintained manually in SHRINE. Also this feature is not used in MedCo for the encrypted terms. It implies though that the ontologies between the different MedCo nodes have to be manually maintained identical at loading time, at least for the ontologies meant to be queried through MedCo.

5.4 MedCo Query Processing

We show in this section a summarized version of the query execution of MedCo, which is implemented as an i2b2 cell that does some pre and post processing around executing the query in the i2b2 data repository. This process is left unchanged from its original implementation described in [23]. It starts at the point where IRCT submits to all the MedCo nodes the query containing the encrypted query terms $\text{ENC}_{Pk_C}[q_v]$.

- **MedCo Distributed Deterministic Tagging**: All the MedCo nodes run together a protocol to compute their own distributed deterministic tag of the encrypted query terms $\text{ENC}_{Pk_C}[q_v]$, using their secret S_i, denoted $\text{DDT}_{S_i}[q_v]$.
- **I2b2 Query Submission**: Each MedCo node queries its local i2b2 instance with the tagged query terms $\text{DDT}_{S_i}[q_v]$, and gets a patient set as a result, which contains both real and dummy patients. The values stored in i2b2 are the tagged values $\text{DDT}_{S_i}[x]$, generated during the data loading. This allows to let i2b2 answers the query as in the normal, insecure process.
- **Dummy Flags Aggregation**: The MedCo nodes retrieve from their local i2b2 instance the patient dummy flags $\text{ENC}_{Pk_C}[f_j]$ of the resulting set. Those flags are a 0 if the patient is a dummy, or a 1 if real, and are encrypted with the collective authority public key Pk_C. They are then homomorphically summed, which gives the encrypted true result of the query $\text{ENC}_{Pk_C}[R_i]$ for the node i.
- **Result Encryption Key Switching** Then the MedCo nodes run a protocol together to switch the encryption from the collective authority key Pk_C to the user's public key Pk_u, the result is $\text{ENC}_{Pk_u}[R_i]$. Optionally the MedCo nodes run a secure shuffling protocol, to break the link between the node and its result. This encrypted result is sent back to IRCT.

5.5 Glowing Bear Result Processing

Following the same process as i2b2 described in section 4.3.4, the encrypted results of the query $\text{ENC}_{Pk_u}[R_i]$ are retrieved from IRCT. The difference here is in the format of the results, as they are encrypted with the public key Pk_u sent along the query.

There is one result $\text{ENC}_{Pk_u}[R_i]$ per MedCo node i. According to access level of the user, it is possible to identify which result is coming from which node, or the result were securely shuffled between the nodes. The format of the result is a JSON string with the following format:

```
{
    "pub_key": "<public key used>",
    "enc_count_result": "<result encrypted with public key>",
    "times": {
        <breakdown of time measurements>
    }
}
```

All the encrypted results $\text{ENC}_{Pk_u}[R_i]$ are then decrypted using the private key pk_u to give R_i, summed together, and displayed to the user using the same process as for i2b2. Additional information containing the more detailed breakdown of results per node is also displayed.

5.5.1 Implementation

A modification is made in the Glowing Bear PIC-SURE results processing to identify correctly when encrypted results are fetched, and extract them. Decryption of the results is done using the cryptography library previously described, and the private key corresponding to the public key used.

A new module is implemented in order to display the detailed breakdown of the MedCo query. It appears as a tab when Glowing Bear detects the presence of MedCo results, and displays the count breakdowns and times measurements from the nodes.

Section 6

System Evaluation

6.1 Requirements Fulfillment

In this section we show how the requirements enumerated section 1.2 are fulfilled.

1. *offer a modern web-based front end for clinical research platforms that allows cohort exploration, based on several types of criteria*
   The chosen front end, Glowing Bear, is a user-friendly web application built using modern technologies (Typescript and Angular). It offers cohort exploration based on the targeted criteria: inclusion and exclusion of ontology query terms.

2. *be compatible with the two major open-source clinical research systems: tranSMART (v17.1) and i2b2*
   It supports both targeted platforms i2b2 and tranSMART 17.1 with the minimal set of features we aimed for.

3. *enable sharing of sensitive data in a privacy-preserving way with MedCo*
   Its support of MedCo enables clinical sites to share sensitive data that could otherwise not be shared without the strong privacy and security guarantees offered.

4. *be easily extensible for future support of additional platforms*
   Integration of the PIC-SURE API through IRCT makes our solution offer a framework for future developers to easily implement support for additional systems, as proposed in the future work section 7.1. Additionally our system has a decoupled architecture making use of implementation-independent RESTful [1] standard APIs. This separation of concerns simplifies development around our system.

5. *alleviate technical hindrances against its use*
   During the design and implementation phases of our solution, many considerations have been taken into account to alleviate technical constraints around its usage:

   (a) *being easy to deploy, even in existing environments*
   Docker [12] is extensively used to both facilitate development and deployment as

detailed in appendix B. A demo version of the whole system can be brought up with a single command.

(b) *not degrading the user experience in existing systems*
Modifications to existing code bases were done in a way that the existing features are not affected, this includes Glowing Bear, IRCT, i2b2, and MedCo.

(c) *enforcing secure authentication*
Implementation of the support for OpenID Connect, a modern standard for secure authentication, was done in all the components of our system. It was implemented from scratch for Glowing Bear, i2b2 and MedCo, and the existing implementation of IRCT was modified to fit our use-case.

(d) *being open-source*
Our system is open-source, not just by the licenses used, but also by our contributions to the community. All the modifications made to the existing i2b2, IRCT, Glowing Bear and MedCo were the subject of *pull requests* on their original repositories, so that they can be considered and potentially mainlined into the original repositories.

(e) *having a practical runtime*
Query times are kept practical. When compared to the query times of the original systems, the overhead is minimal. In tranSMART's case it stays the same as the Glowing Bear implementation remains. In the i2b2 case there is the overhead of using IRCT, however this overhead is minimal and close to a constant as the only processing is the translation of the query. This translation process is close to a simple mapping and is in the order of a dozen milliseconds, not including I/O. In the MedCo case the time is actually reduced, as the query do not pass through SHRINE and is sent directly to the MedCo nodes.

6.2 Limitations

6.2.1 Combining Data Sources

Queries made in our system cannot combine query terms from different data sources at the same time. While simply combining query terms is straightforward to implement on a purely technical level, querying different clinical research platforms at the same time implies harmonizing the semantic of their data. This could be done either by enforcing the same semantic at the loading time, or having a mapping mechanism. In both cases, this was left out of the focus of this thesis.

6.2.2 tranSMART Support in the Interoperability Layer

Support of tranSMART in the interoperability layer is implemented directly in Glowing Bear, while support for other platforms is through the PIC-SURE API. The cause of this situation is the preexisting support tranSMART in Glowing Bear, which has some advanced features. While it was feasible to re-implement support of tranSMART through PIC-SURE, the immediate added value of this work would have been low. This however may raise concerns for the future development around our system because of the increased complexity.

6.2.3 Contributions to the Open-Source Community

The modifications or new implementations made in existing open-source projects were all the subject of *pull requests* on their original repositories, in order to mainline our work. There is however no guarantee that the maintainers of those projects will accept them. If they are not accepted, either they can be with additional modifications, or they are simply rejected. In the first case, additional implementation effort to mainline our changes would be needed, but there would not be further consequences if properly done. In the second case, we would need to maintain a fork of the original code bases and regularly integrate the new developments back into the fork, which would increase the complexity of maintaining our system.

6.2.4 Features Offered through PIC-SURE

The cohort exploration features offered in Glowing Bear when using PIC-SURE remain basic: inclusion and exclusion of ontology query terms. While this is a functional proof of concept, it is likely that clinical sites desire more advanced features, such as the ones available in Glowing Bear when using tranSMART. This implies that additional efforts will be needed in the future on the system to implement support of more advanced features, in order to convince potential clinical sites to adopt our system.

Section 7

Conclusion

Throughout this thesis we designed and implemented a system providing a front end able to query the clinical research systems *i2b2*, *tranSMART* and *MedCo*, with negligible overhead compared to the original platforms. The supported queries allow researchers to do cohort exploration based on common criteria, giving technical means to clinical sites to share their data.

By this we are alleviating the challenges stated in the introduction. First the technical challenge, we are bringing together technically the two main open-source clinical research systems with an interoperability layer that can be used by a unique front end. Second the legal and ethical challenge, by integrating MedCo in this interoperability layer, clinical sites are given the option to securely share their data with strong privacy and security guarantees.

All in all, with our system we took a step towards the technical convergence of the main clinical research systems. We are providing researchers with a modern a powerful front end that abstracts away the technical differences between platforms, and enable clinical sites to share that would otherwise not be shared without the strong privacy and security guarantees offered.

7.1 Future Work

Future work on our system could be done on both the front end and the interoperability layer. On the front end features that exists for tranSMART could be added for the PIC-SURE implementation, such as a standard mechanism for saving queries, data export, or advanced data analysis. Glowing Bear and the interoperability layer could be easily extended to support more systems with low implementation effort, systems such as HAIL [30], SciDB [5] or SHRINE [4]. Those are examples of systems partially supported by PIC-SURE, but who need adaptations to be fully integrated. To bring more uniformity and easier maintenance, the support of tranSMART could be brought through PIC-SURE. We could also imagine a way to bring uniformity in the semantic data coming from clinical research systems, allowing to query data from multiple systems at a time.

A work already in progress on our solution consists in bringing the support of HAIL through Livy [26] in IRCT, and combining it with i2b2 for powerful cohort exploration with genomic data.

References

[1] Roy Fielding. “Representational state transfer”. In: *Architectural Styles and the Design of Netowork-based Software Architecture* (2000), pp. 76–85.

[2] Centers for Disease Control, Prevention, et al. “HIPAA privacy rule and public health. Guidance from CDC and the US Department of Health and Human Services”. In: *MMWR: Morbidity and mortality weekly report* 52.Suppl. 1 (2003), pp. 1–17.

[3] Gerhard Nahler. “ICD-9 code”. In: *Dictionary of Pharmaceutical Medicine*. Springer, 2009, pp. 88–88.

[4] Griffin M Weber et al. “The Shared Health Research Information Network (SHRINE): a prototype federated query tool for clinical data repositories”. In: *Journal of the American Medical Informatics Association* 16.5 (2009), pp. 624–630.

[5] Paul G Brown. “Overview of SciDB: large scale array storage, processing and analysis”. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM. 2010, pp. 963–968.

[6] Shawn N Murphy et al. “Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2)”. In: *Journal of the American Medical Informatics Association* 17.2 (2010), pp. 124–130.

[7] Anne Van Kesteren et al. “Cross-origin resource sharing”. In: *W3C Working Draft WD-cors-20100727, latest version available at< http://www. w3. org/TR/cors* (2010).

[8] Mohamed Omar Rayes. “One-time password”. In: *Encyclopedia of Cryptography and Security* (2011), pp. 885–887.

[9] Dick Hardt. *The OAuth 2.0 authorization framework*. Tech. rep. 2012.

[10] Michael Jones and Dick Hardt. *The oauth 2.0 authorization framework: Bearer token usage*. Tech. rep. 2012.

[11] Gavin Bierman, Martin Abadi, and Mads Torgersen. “Understanding typescript”. In: *European Conference on Object-Oriented Programming*. Springer. 2014, pp. 257–281.

[12] Dirk Merkel. “Docker: lightweight linux containers for consistent development and deployment”. In: *Linux Journal* 2014.239 (2014), p. 2.

[13] Nat Sakimura et al. “OpenID Connect Core 1.0 incorporating errata set 1”. In: *The OpenID Foundation, specification* (2014).

[14] Elisabeth Scheufele et al. “tranSMART: an open source knowledge management and high content data analytics platform”. In: *AMIA Summits on Translational Science Proceedings* 2014 (2014), p. 96.

[15] Michael Jones. *JSON web key (JWK).* Tech. rep. 2015.

[16] Michael Jones, John Bradley, and Nat Sakimura. *JSON web signature (JWS).* Tech. rep. 2015.

[17] Michael Jones, John Bradley, and Nat Sakimura. *Json web token (jwt).* Tech. rep. 2015.

[18] Ewa Syta et al. "Certificate cothority: Towards trustworthy collective cas". In: *Hot Topics in Privacy Enhancing Technologies (HotPETs)* 7 (2015).

[19] Chirag J Patel et al. "A database of human exposomes and phenomes from the US National Health and Nutrition Examination Survey". In: *Scientific data* 3 (2016), p. 160096.

[20] Alex AT Bui, John Darrell Van Horn, NIH BD2K Centers Consortium, et al. "Envisioning the future of big data biomedicine". In: *Journal of biomedical informatics* 69 (2017), pp. 115–117.

[21] Serena Scollen, Angela Page, Julia Wilson, et al. "From the Data on Many, Precision Medicine for One: The Case for Widespread Genomic Data Sharing". In: *Biomedicine Hub* 2.Suppl. 1 (2017), pp. 15–15.

[22] A. Oehmichen et al. "A Multi Tenant Computational Platform for Translational Medicine". In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS).* July 2018, pp. 1553–1556. DOI: 10.1109/ICDCS.2018.00167.

[23] J. L. Raisaro et al. "MedCo: Enabling Secure and Privacy-Preserving Exploration of Distributed Clinical and Genomic Data". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (2018), pp. 1–1. ISSN: 1545-5963. DOI: 10.1109/TCBB.2018.2854776.

[24] *2018 reform of EU data protection rules.* URL: https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en.

[25] *Apache Axis2/Java - Next Generation Web Services.* URL: http://axis.apache.org/axis2/java/core/.

[26] *Apache Livy: A REST Service for Apache Spark.* URL: https://livy.incubator.apache.org/.

[27] *Glowing Bear: Modern cohort selector for i2b2 tranSMART.* URL: https://glowingbear.app.

[28] *Glowing Bear Source Code.* URL: https://github.com/thehyve/glowing-bear.

[29] *GopherJS - A compiler from Go to JavaScript.* URL: https://github.com/gopherjs/gopherjs.

[30] *Hail: an open-source, scalable framework for exploring and analyzing genomic data.* URL: https://hail.is/index.html.

[31] *Hibernate Framework.* URL: https://en.wikipedia.org/wiki/Hibernate_(framework).

[32] *i2b2 community.* URL: https://community.i2b2.org/.

[33] *i2b2 Webclient Source Code.* URL: https://github.com/i2b2/i2b2-webclient.

[34] *i2b2 Workbench Source Code.* URL: https://github.com/i2b2/i2b2-workbench.

[35] *IRCT Source Code.* URL: https://github.com/hms-dbmi/IRCT.

[36] *Keycloak: Open Source Identity and Access Management.* URL: https://www.keycloak.org/.

[37] *npm: the package manager for javascript.* URL: https://www.npmjs.com/.

[38] *Patient-centered Information Commons: Standardized Unification of Research Elements.* URL: https://pic-sure.org/.

[39] *PIC-SURE RESTful API Documentation.* URL: http://bd2k-picsure.hms.harvard.edu.

[40] *The Apache Groovy programming language.* URL: http://groovy-lang.org/.

[41] *The Go Programming Language.* URL: https://golang.org/.

[42] *TranSMART Community.* URL: http://transmartfoundation.org/community/.

[43] *tranSMARTApp Source Code.* URL: https://github.com/transmart/transmartApp.

[44] *Web Application Archive (WAR).* URL: https://en.wikipedia.org/wiki/WAR_(file_format).

[45] *YUI Library.* URL: https://yuilibrary.com/.

Appendix A

Online Resources

This appendix lists the online resources where the result of our work can be found.

A.1 Git Repositories

- `glowing-bear-backend` repository, branch `fork/thehyve`
  `https://github.com/thehyve/glowing-bear-backend`
  - IRCT fork, modified for the need of our project
  - Docker-based deployment scripts
- `glowing-bear` repository, branch `picsure`
  `https://github.com/thehyve/glowing-bear`
  - Glowing Bear fork, modified with support of PIC-SURE API and OpenID Connect
- `i2b2-core-server` repository, branch `oidc-authentication`
  `https://github.com/thehyve/i2b2-core-server`
  - i2b2 server fork, modified with support of OpenID Connect
- `unlynx-crypto-js-lib` repository
  `https://github.com/lca1/unlynx-crypto-js-lib`
  - javascript library for the MedCo cryptographic primitives
  - source Go code of the library
- `medco-deployment`, branch `shrineremoved`
  `https://c4science.ch/source/medco-deployment`
  - Docker-based deployment of MedCo, in a version without SHRINE

A.2 Other Resources

- Docker Cloud repository, `medco` organization
  `https://cloud.docker.com/swarm/medco`
  - `i2b2` docker image
  - `i2b2-medco` docker image
- NPM repository, `medco` organization
  `https://www.npmjs.com/package/@medco/unlynx-crypto-js-lib`
  - NPM package of the javascript MedCo crypto library

Appendix B

Docker-Based Deployment

This appendix describes the docker-based infrastructure put in place to test and deploy the whole system.

B.1 Docker Images

B.1.1 i2b2

- Image name: `i2b2`
- Base image: `wildfly`
- Ports exposed
  - 8080: deployments endpoint
  - 9990: WildFly management interface (default credentials: `admin/admin`)

This sets up a working instance of i2b2 at the URL `/i2b2/services/`, with the i2b2 demo data loaded. Some patches are applied during building, that include the support of OpenID Connect authentication. The instance is based on Apache Axis2 with the following services corresponding to i2b2 cells:

- `CRC`: Clinical Research Chart (data repository)
- `ONT`: ONTology management
- `PM`: Project Management (authentication and authorization)
- `WORK`: WORKflow management (query, result sharing)
- `FR`: File Repository
- `IM`: Identity Management

B.1.2 i2b2 MedCo

- Image name: `i2b2-medco`
- Base image: `i2b2`
- Ports exposed
  - 8080: deployments endpoint
  - 9990: WildFly management interface (default credentials: `admin/admin`)

This sets up a working instance of an MedCo node, based on an i2b2 instance, at the URL `/i2b2/services/`, with the i2b2 demo data loaded, and the MedCo database structure. On top of the i2b2 cells previously described, it adds the `MedCo` cell.

B.1.3 IRCT

- Image name: `irct`
- Base image: `wildfly`
- Ports exposed
  - 8080: deployments endpoint
  - 9990: WildFly management interface (default credentials: `admin/admin`)
- Volumes
  - `/irct-src/`: sources of IRCT
  - `/opt/jboss/.m2/repository/`: Maven packages cache repository

This sets up a working instance of IRCT at the URL `/IRCT-CL/`, with the following data sources pre-configured:

- `i2b2-local`: local instance of i2b2 using fixed credentials
- `i2b2-local-jwt`: local instance of i2b2 using OpenID Connect authentication
- `i2b2-public`: public test instance of i2b2 using fixed credentials
- `i2b2-medco-local`: local instance of MedCo using fixed credentials
- `i2b2-medco-local-jwt`: local instance of MedCo using OpenID Connect authentication

The starting script of the image will load the data in the PostgreSQL database if not already present, and will always compile the latest version of the IRCT source code.

B.1.4 PostgreSQL Database Server

- Base image: `postgres`
- Ports exposed
  - 5432: PostgreSQL port
- Volumes
  - `/var/lib/postgresql/data/`: PostgreSQL database files

This sets up a working PostgreSQL server and loads the database structure needed for the other images. The structure is a database and a corresponding user for `irct` and `i2b2`. The actual data is loaded at the first initialization of the other images.

B.1.5 Lighttpd Web Server

- Image name: `lighttpd`
- Base image: `debian`
- Ports exposed
  - 80: HTTP port

This sets up a working Lighttpd server with PHP and install several services:

- `/phppgadmin/`: phpPgAdmin PostgreSQL management tool (default credentials: `postgres/postgres`)
- `/i2b2-client/`: the i2b2 webclient, using the local i2b2 instance (default credentials: `demo/demouser`)
- `/i2b2-admin/`: the i2b2 admin tool, managing the local i2b2 instance (default credentials: `i2b2/demouser`)

B.2 Docker-Compose Run Configuration

Two *docker-compose* configuration profiles are provided as example:

- `docker-compose.local.yml`: deploys a local development environment with all the components.
- `docker-compose.test.yml`: deploys a remote test environment; needs to be adapted to the server on which it is deployed.