Design of a Service-Oriented Architecture for Federated Systems

Daniel Calegari dcalegar@fing.edu.uy
Marcos Viera mviera@fing.edu.uy
Regina Motz rmotz@fing.edu.uy

Instituto de Computación, Facultad de Ingeniería
Universidad de la República
Julio Herrera y Reissig 565, 5to piso
11300 Montevideo, Uruguay
Fax: (+598) (2) 7110469

Abstract

A Federated System is a collection of independent, cooperative, possibly heterogeneous and autonomous computer systems (usually database systems) which allows sharing all or some of its data. A Service-oriented Architecture is an application architecture whose functionalities are defined as independent services which offer transparent communication between physically distributed components, possibly heterogeneous and autonomous. In this context, it is interesting to analyze how a Federated System can be designed within the ideas proposed by Service-Oriented Architectures. This paper presents the design of a Service-Oriented Architecture for Federated Systems. The architecture supports many users sharing data; access control to the data based on access rights which generates many views from a data source, as well as allowing a high automation level for the integration and querying processes. In addition, the bases of a federation's management framework are defined. This framework, as well as the architecture, is validated through an evolutionary prototype towards a completely functional implementation.

Keywords: Federated Systems, Service-Oriented Architecture, Databases, Distributed Systems

Workshop: Workshop de Ingeniería de Software y Bases de Datos

1. Introduction

Federated Systems come into the game as an answer to the growing needs of cooperation between independent systems in order to share data and provide new functionalities to the users.

A Federated Database System [1] coordinates the cooperation between data sources (originally databases, but it could be extended to information systems, among others) to provide a unified view of them to different users. Three properties characterize a Federated System:

- **Distribution**: The data of a federation is distributed across different sources. This distribution, which is not only logical but also physical, generates the need of having flexible mechanisms of integration and remote communication to connect the federation with the sources.
- **Autonomy**: Each source is independent and decides its participation level within the federation. As a consequence, the sources must be loosely coupled with the federation, in order to be able to execute independently and decide the access level to the data for each user.
- **Heterogeneity**: There can be different kinds of heterogeneity between sources: in the platform they execute, in its semantics, in its structure, in the query language, among others. So, it's convenient to define a unified connectivity mechanism in the format of the schemas, in the query language used in the interaction with the federation and in the connectivity technology used.

The five-level architecture proposed in [1] presents the logical structure of a Federated System considering the properties previously mentioned. Each source has a **local schema** in its own language (also each source has its own query language). This schema is transformed by a **transforming processor** into a schema with a canonical representation for all the federation (also transforms the queries in the canonical language of the federation to the language of the source), generating a **component schema**. Each source defines the piece of information to which the federation will be able to access. Based on this access control a **filtering processor** generates an **export schema**. From the export schemas of each source, the federation, by means of a **constructing processor**, generates the **federated schema**. After another access control, performed by a filtering processor, this schema is transformed into an **external schema**, to which the users access.

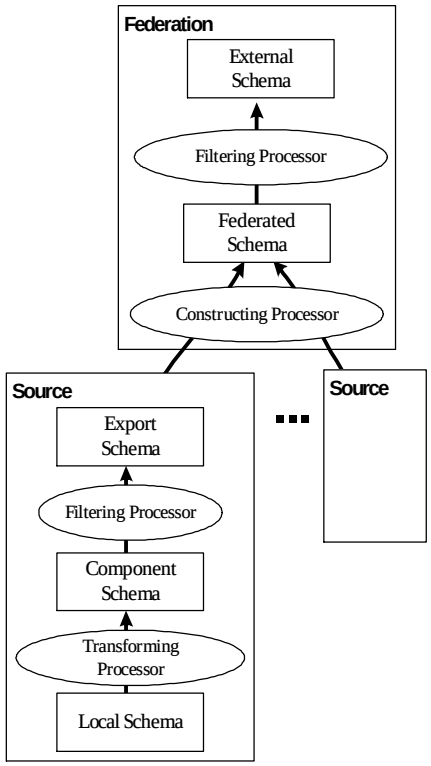

Figure 1: Five-level Architecture for Federated Systems

Except for the transforming processor and the local schema, the rest does neither depend on the nature of each source nor on the federation itself. This motivates the construction of a generic solution in order to provide support to federation's management.

On the other hand, there is a clear separation between the responsibilities of the sources and the federation. It is possible to see the sources as service suppliers, offered in a uniform way by all of them, and the federation as a consumer of these services.

A Service-Oriented Architecture [2] is an application architecture where the functionalities are defined as independent services, with well defined interfaces, which can be called in given sequences to build business processes. This architecture provides a framework that allows heterogeneity, integration and reusability of the participant components in a flexible environment.

In this context, it is interesting to analyze how a Federated System can be designed within the ideas proposed by the Service-Oriented Architectures.

The primary goal of this paper is to present the design of a Federated Systems architecture exploring the ideas proposed by the Service-Oriented Architectures. The design contemplates a Federated System with support for multiple users and access control to the data, based in access rights that generate multiple views for a source. A second objective is to define the basis of a generic solution (*framework*) in order to give support to federation's management and to explore possible levels of automation. Also, the implementation of the framework validates the architecture as it evolves towards a completely functional implementation.

The rest of this paper is organized as follows. In Section 2 the basic capabilities of the proposed architecture are exposed through a case study. In Section 3 the design of a Service-Oriented Architecture for Federated Systems is presented. In Section 4 the basis of a generic solution are analyzed and the prototype is presented. Finally, in Section 5 the final conclusions and future work guidelines are discussed.

2. Capabilities

In order to understand the architectural design of the federated system we will first introduce the capabilities that the solution must have with a Medical Federation case study. This federation is composed by many hospitals geographically distributed, each one with its own computer system. The objective of the federation is to integrate every computer system providing a unified view of the data concerning the clinical histories of their patients.

The data the sources use are structurally and semantically heterogeneous. Moreover, each source can handle different information. There also exists heterogeneity given by the platform in which each source runs. There are three sources: a relational database, an object-oriented system and a XML-files source. Each source must have a local schema to be part of the federation, so each one must define its own data schema. In the relational database the local schema is explicitly defined (it is shown by its tabled structure) but in other cases as the object-oriented system or the XML files there is no explicit data representation. For this reason, the owner of each source must infer the local schema, implicit in the data that the source uses. The local schema of each source in the case study is shown in Figure 2. Each schema is presented with a different notation (Database tables, UML Class Diagram [3] and Entity-Relationship Model [4]) to point out the heterogeneity of the sources.

As a first step in order to generate the federation, each source must define which data will be exported, that is the component schema. Considering the objective of the Medical Federation, each source defines its own component schema (according to the participation level that they will have in the federation) as shown in Figure 3. The component schemas are presented with a canonical notation, a UML Class Diagram.

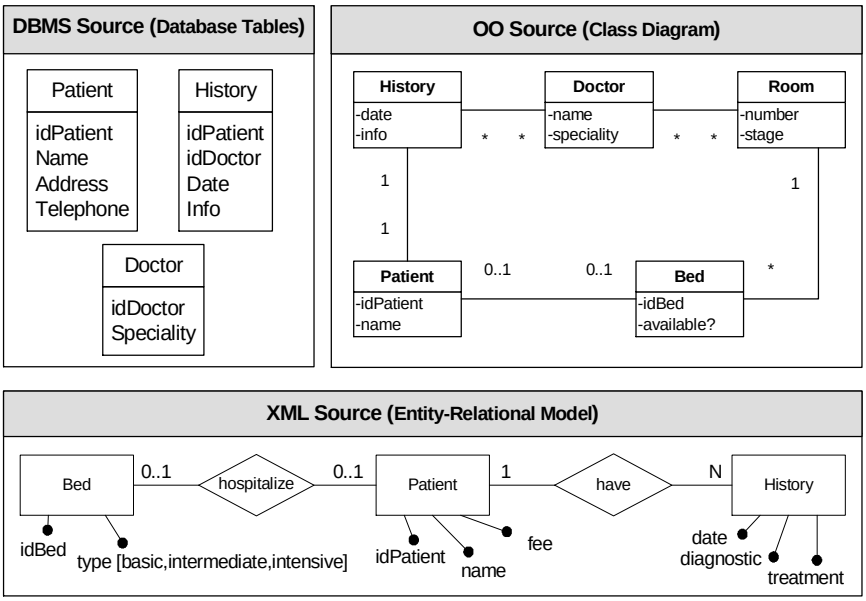

Figure 2: Case Study Sources

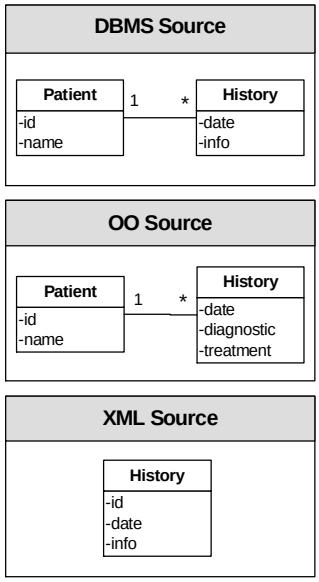

Figure 3: Case Study Export Schemas

The federated schema that will be generated depends on the strategy used. For example, the strategy followed could maximize the information extracted from the sources and therefore it will generate a schema that contains all the available information. Using this strategy in the case study, the schema shown in Figure 4 is generated. In this schema, it is possible to query the medical history of a patient. This query will be answered by every source. However, as a consequence of this strategy, there could be queries in which a source doesn't contain enough information to answer. Each federation must define a policy to clearly state what happens in this case. Some federations could decide if they will cancel the entire query or just to make the query in the sources that can answer.

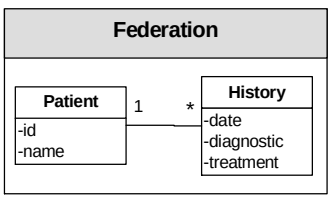

Figure 4: Case Study Federated Schema

A desired capability is to let sources define access rights over its data and let the federation define user profiles that will use these rights. The following sections present some extensions to the export schema and the filtering process of the sources to support these capabilities.

2.1. Access Rights

Through the definition of access rights, a source defines the participation level it will have in the federation, not only defining the exports the schema to share but also the access rights who have the federation over each schema element. For this purpose, the export schema must be extended to include *read/write* access rights in each schema element: classes, attributes and relationships.

Figure 5 shows the access rights defined in the case study (for simplicity there are rights only at class level). The federated schema is the same as the one of the *OO Source*. In this case, it is valid to query and modify clinical histories but not to delete patients because they have only *read* rights.

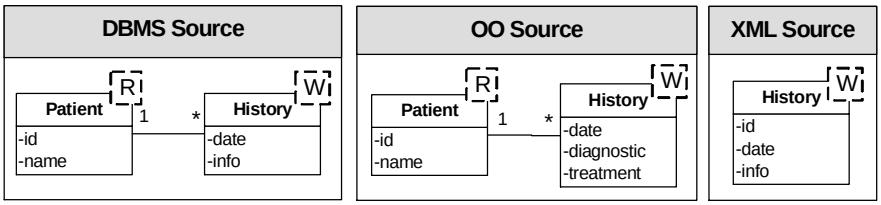

Figure 5: Export Schemas with Access Rights (R-*Read*, W-*Write*)

The access control to the data can be complex. It is convenient to define it together with the integration strategy since it is necessary to define, among other things, what happens when two semantically similar elements have different access rights.

The filtering processors of the source and federation are responsible for checking the access rights of each *CRUD* operation over the external schema. At this point, two actions can be taken: discard a wrong query or carry it out only where it has enough access rights. In the second case, the query must be transformed into a partial query and the results can be unexpected. This problem is fairly common, mainly when creating/updating data into the federation, but it is out of the scope of this work.

2.2. Profiles

The idea of access rights is extended by the definition of user profiles, generating different "views" of the federation. In this way, each source can export more than one schema, one for each profile with its own access rights.

The federation shown in the case study can be part of a *doctor* profile. It is now possible to create a new profile, called *hospitalization advisor*, for consulting bed-availability in hospitals. For this new profile the schemas generated are shown in Figure 6 (the *DBMS Source* decides not to join the federation for this profile).

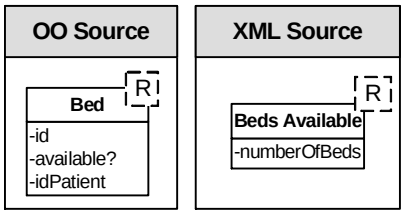

Figure 6: Export Schema for the Hospitalization Advisor Profile

Profiles can represent many actors inside a federation (like the doctor and the hospitalization advisor) as different federations. This lets each source participate in more than one federation with the same mechanism and also supporting more than one user in one federation, each one with different access rights and as a consequence with different external schemas. In order to support this capability it is necessary to verify the user identity together with the access rights in each transforming processor.

3. Proposed Architecture

A service-oriented architecture formed by two components is proposed, one component on the federation side and the other one on the sources' side. The following diagram shows the modules within each component and their dependencies.

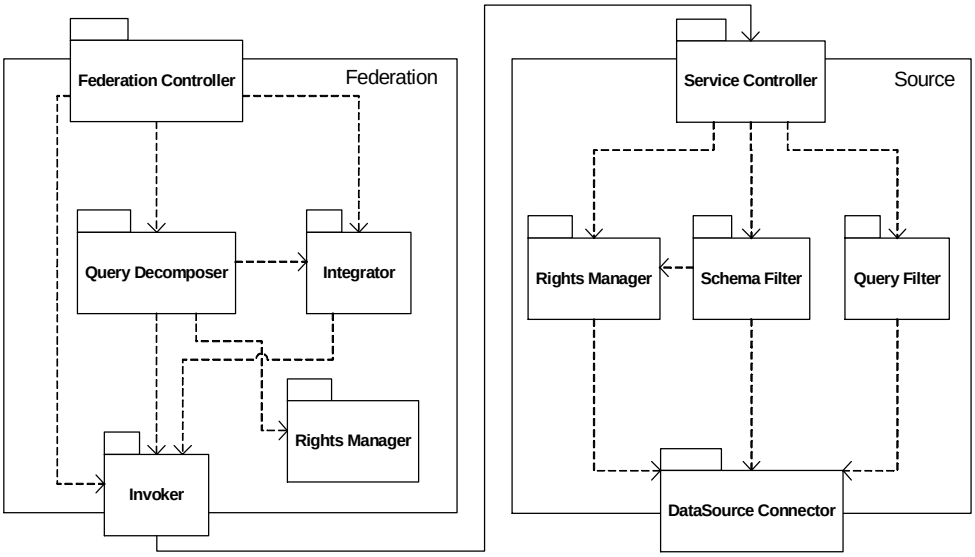

Figure 7: Designed Architecture

The following description shows the responsibilities of each module.

Federation	
Federation Controller	Works as an intermediary between the user and the federation. Allows the user to interact at two levels: at an administrative level to generate the federation on the basis of different sources and from a user level to work on the federated schema.
Integrator	Generates the federated schema based on the schemas that arrive from each source and integrates the access rights.
Query Decomposer	Allows the query decomposition of a query made by the user over the federation into specific queries to each one of the sources. Requires information from the *Integrator* to decompose each query. Verifies the user's rights over each source before making the query. In addition, it is responsible of constructing the result of a distributed query.
Invoker	Knows the location of each source and invokes its services by their *Service Controller*.
Rights Manager	Verifies the access rights that a user has over a schema.

Source	
Service Controller	Exports the services of a source. Receives orders of the federation and delegates them to the responsible component within the source
Query Filter	Receives a query, verifies the rights of the user who had made the query (using the *Rights Manager*) and executes the query on the source.
Schema Filter	Generates and controls the exportation schemas of a source, based on the profiles and permissions defined for each schema.
DataSource Connector	Translates the model and query language of each source to the canonical model and language of the federation.
Rights Manager	It's the same module described in the component *Federation*.

The mapping between the components defined in the proposed architecture and the components defined in the five-tier architecture [1] are shown in Figure 8.

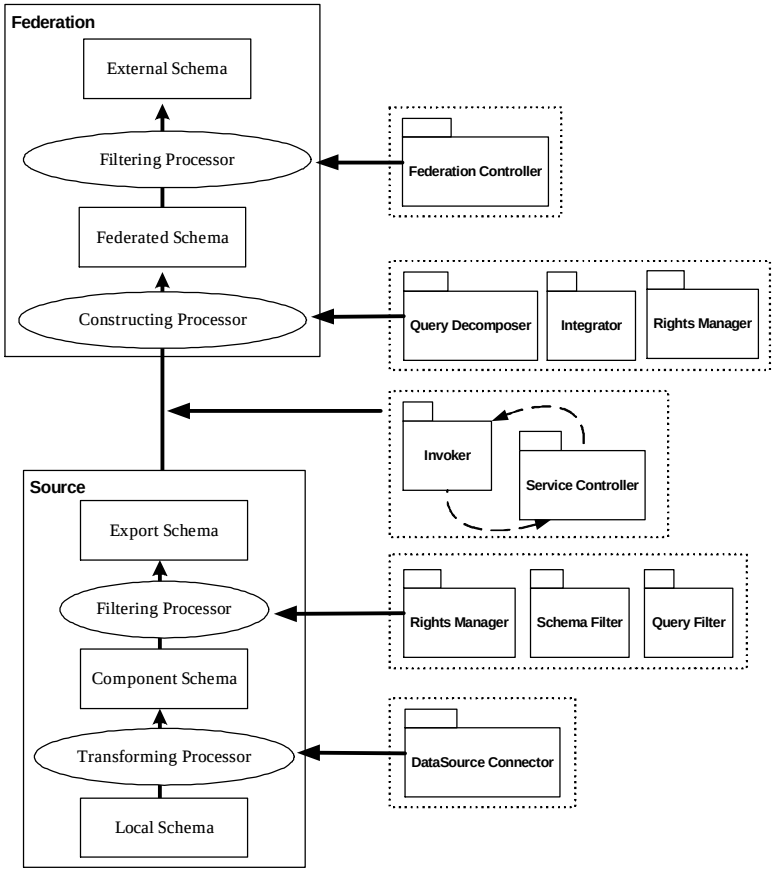

Figure 8: Designed Architecture - Five-level Architecture Mapping

3.1. Services

The federation offers administrative (sources, profiles and permissions management) and federation (schema integration and *CRUD*[1] operations on the integrated schemas) functionalities. In order to fulfill these functionalities the services offered by each source are used, which are: schema and permissions for profile exportation, and *CRUD* operations. The interactions between components in order to offer and consume these services are shown in the following figures.

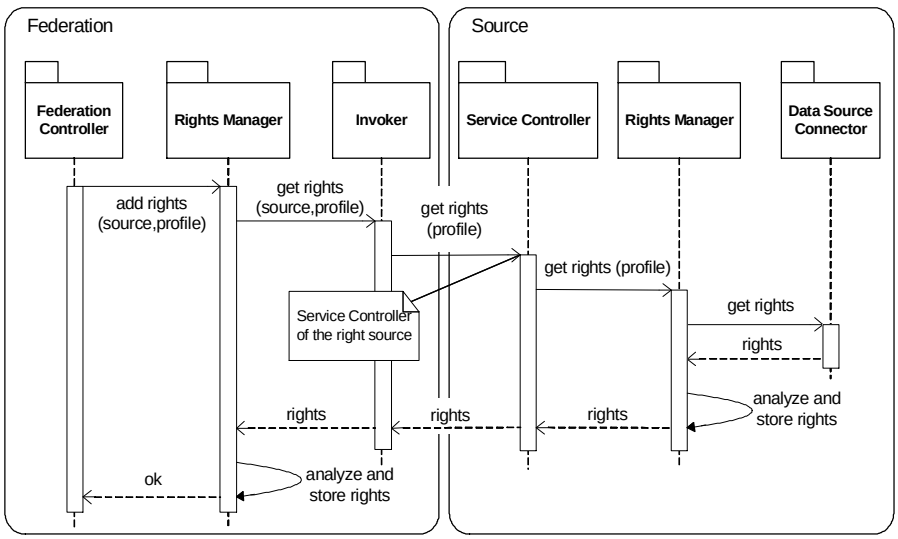

Figure 9: Rights Exportation Interaction

[1] *CRUD* (*Create*, *Retrieve*, *Update* y *Delete*)

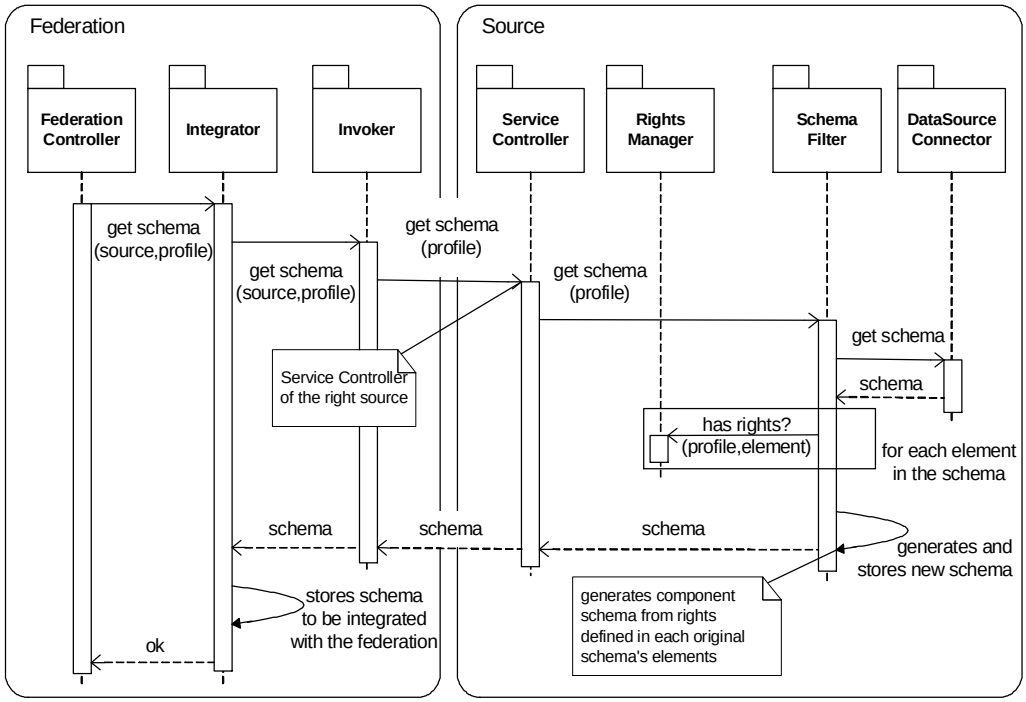

Figure 10: Schema Exportation Interaction

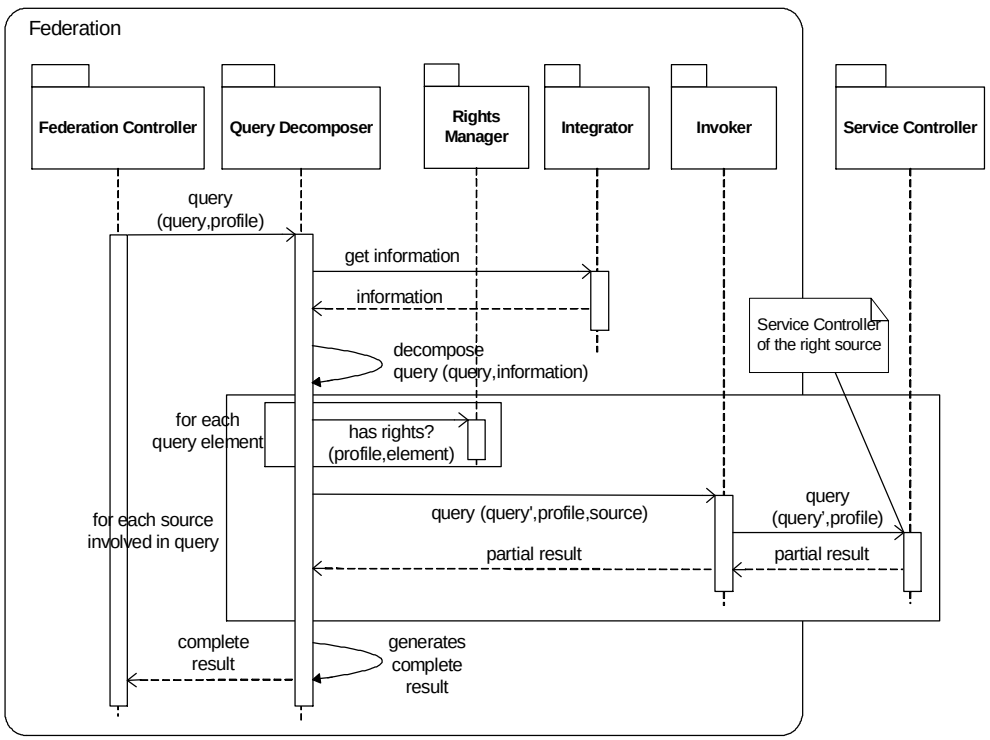

Figure 11: Query Interaction (Federation side)

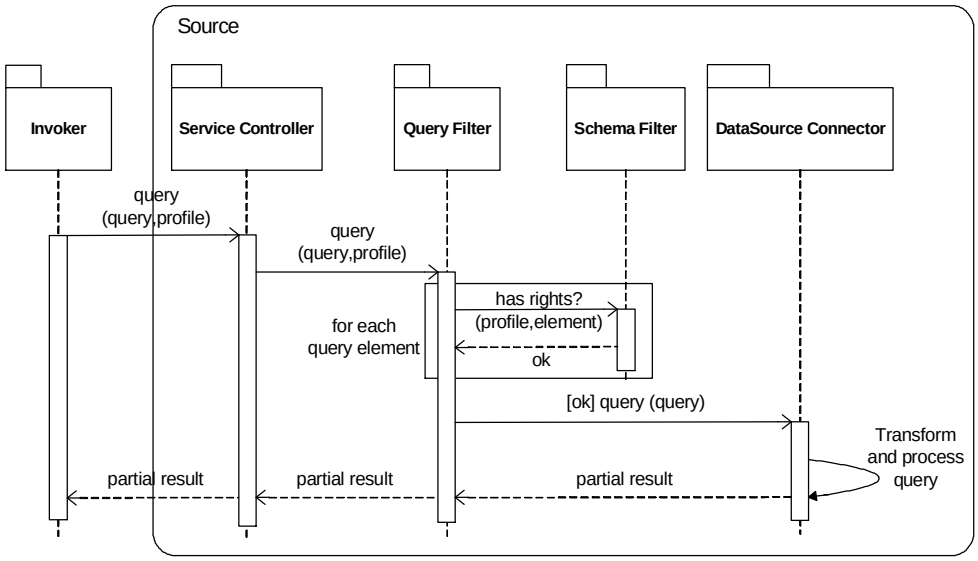

Figure 12: Query Interaction (Source side)

3.2. Automation

During the execution of a great part of the functionalities offered by the federation, interaction with an external user is not required. This allows the (semi) automation of the integration, query, data maintenance, and federation management processes.

If a connection mechanism is chosen, the components that implement it (*Invoker* and *Service Controller*) can be automated. By fixing a format of the export schema (canonical model, query language, permissions and profiles) the components *Rights Manager* and *Schema Filter* can be automated. Finally, by choosing an automatic or semi-automatic schema integration strategy (for example SIM[2] [5][6]), and defining, based on it, permission integration and query decomposition strategies, *Integrator*, *Query Decomposer* and *Query Filter* are automated. A hardly automatable component, because it depends on the nature of each source, is the *DataSource Connector*, although the transformations of schemas and queries for certain types of sources could be studied.

4. Framework

Most of the architecture doesn't depend on specific features of each federation. This particularity allows us to generate a federation's management framework. As we saw in the last section, to instantiate this framework we have to define the connectivity mechanism, a concrete export schema and the different strategies involved. This is what we will see in this section.

4.1. Connectivity Mechanism

Web Services [2] is the most used connectivity technology for service-oriented architectures. *Web Services* are modular applications, self-described, self-contained which can be accessed through a network. Based in open standards, they allow the construction of web applications using any platform, object model and programming language. *Web Services* modularity and flexibility [7] make them appropriate for applications' integration with a minimum programming effort. This is why we propose the use of *Web Services* as the connectivity mechanism between the federation and the sources.

[2] SIM strategy is a database federation system's integration strategy based on correspondences between sub-schemas of the schemas to be integrated. It is declarative and semiautomatic.

4.2. Export Schema

As a canonical model, it is possible to use ODMG [8] because its expressive power allows it to cover different data models [9]; it offers an object description language (ODL) and an object query language (OQL) on the model. OQL, an extension of SQL, doesn't have operators for Create, Update, and Delete object instances. In this case we use SQL operators.

Each source must create its component schema using ODL. ODL will be the specification language of the export, federated and external schema. As an example, the federated schema of the case study for "doctor" profile is shown in Figure 13.

Federation

```
interface Patient (extent patients key id){
  attribute Integer id;
  attribute String name;
  relationship Set<History> have inverse History::of;
};
interface History (extent histories){
  attribute Date date;
  attribute String diagnostic;
  attribute String treatment;
  relationship Patient of inverse Patient::have;
};
```

Figure 13: Federation ODL

Queries on the external schema are carried out in OQL and then decomposed in sub-queries, one for each source. An example of global query, query decomposition and result format for the case study is shown in Figure 14.

Federation

```
select h.date, h.diagnostic, h.treatment
from Patient p, p.have h
where p.id = ID
```

XML Source

```
select h.date, h.info
from History h
where h.id = ID
```

DBMS Source

```
select h.date, h.info
from Patient p, p.have h
where p.id = ID
```

OO Source

```
select h.date, h.diagnostic, h.treatment
from Patient p, p.have h
where p.id = ID
```

Result

```
bag(History(date:'Date',diagnostic:"String",treatment:"String"),...)
Date is: struct(month: Integer, day: Integer, year: Integer)
```

Figure 14: Query

Profiles and their access rights are specified using XML format [10] whose structure is defined in a DTD [11] as shows Figure 15. An example of a profile for the *XML Source* of the case study is shown in Figure 16. This example shows the "doctor" profile with *read/write* rights over "History" class and its attributes, while the "hospitalization advisor" profile has only *read* rights over "BedsAvailable" class and its attribute.

```
<!ELEMENT Rights (Right+)>
<!ELEMENT Right (R,RW)>
<!ATTLIST Right profile CDATA #REQUIRED>

<!ELEMENT R EMPTY>
<!ATTLIST R name CDATA #REQUIRED
   type (Class|Attribute|Relationship) #REQUIRED>

<!ELEMENT RW EMPTY>
<!ATTLIST RW name CDATA #REQUIRED
   type (Class|Attribute|Relationship) #REQUIRED>
```

Figure 15: Access Rights DTD

XML Source

```
<Rights>
  <Right profile="doctor">
    <RW name="History" type="Class" />
    <RW name="History::id" type="Attribute" />
    <RW name="History::date" type="Attribute" />
    <RW name="History::info" type="Attribute" />
  </Right>

  <Right profile="advisor">
    <R name="BedsAvailable" type="Class" />
    <R name="BedsAvailable::numberOfBeds"
       type="Attribute" />
  </Right>
</Rights>
```

Figure 16: XMl Source Access Rights

4.3. Strategies

ODMG allows using into the framework *SIM* methodology. Access rights integration (and inconsistency policy), decomposition and query validation strategies depends on it. The concrete definition of this kind of strategies is beyond the scope of this work.

4.4. Prototype

A prototype[3] of the framework has been developed in order to validate the architecture. The prototype has a full implementation of the architecture structure and services explained in section 3. Also, some processes have been automated. it contains a functional version of *Federation Controller*, *Invoker* and *Rights Manager* component in the federation; and *WebService Controller*, *Schema Filter* and *Rights Manager* component in the source.

The federation management and the access rights validation on each schema component have been automated. The access rights validation uses only rights defined by each source, not considering integrated rights (this depends on the integration strategy as we saw). Integration, query and data maintenance processes have not been automated. As a consequence, the prototype has been adapted to support just some static *CRUD* queries.

The prototype showed the feasibility of the proposed architecture as it consists the first version of a functional system. It also helped in discovering future work required to build a completely functional version, as will be exposed in the next section.

5. Conclusions and Future Work

This paper presents the design of a Service-Oriented Architecture for Federated Systems. The designed architecture, based on the five-tier architecture in Section 1, supports all of the characteristic properties of Federated Systems: distribution, autonomy and heterogeneity of sources, as well as some capabilities like user profiles, access rights to federated data for each profile and source transparency. Moreover, it enables the integration of not only databases but also applications in an abstract manner for the user because it can see the services but not the provider. The paper analyzes the possible automation level of each component within the architecture and defines the basis of a federation's management framework using these ideas. The framework, as well as the architecture, was validated through an evolutionary prototype.

There are works which analyze the application of Service-Oriented Architectures for Federated Systems. In [13], an architecture for federation's management is built. In this case, the architecture is not context-free and its focus is the high-level administrative services of the federation, not the integration and querying processes this paper focuses on.

[3] The prototype was implemented with *J2EE Platform* [12]; its source code is available at: http://www.fing.edu.uy/inco/grupos/coal/investigacion/publicaciones/

Some future work is required in order to continue the evolution of the framework's prototype towards a completely functional version:

- Use of heterogeneity to add new source's and federation's components in different languages in order to generate a multi-language framework.
- Incorporation of new and better federation's management tools as those in [13].
- Incorporation of concrete schema integration and query decomposition (and validation) strategies as *SIM*. This promotes the study of new strategies with access rights support.
- Study of alternatives for the instantiation of the framework. This means the study of different schema formats (for example UML Class Diagrams and not ODMG) and query languages (for example OCL [3] and not OQL) in order to make a comparative analysis between framework's instances.

6. References

[1] A. Sheth, J. Larson, *Federated database systems for managing distributed, heterogeneous, and autonomous databases*. ACM Comput. Surv. 22, 3 (September 1990), 183-236.

[2] D. Barry, *Web Services and Service-Oriented Architectures: The Savvy Manager's Guide*, Morgan Kaufmann Publishers, 1 ed., 2003, ISBN 1-55860-906-7.

[3] Object Management Group. *OMG Unified Modeling Language Specification*, Version 1.5, March 2003.

[4] P. Chen, *The entity-relationship model—toward a unified view of data*. ACM Trans. Database Syst. 1, 1 (March 1976), 9-36.

[5] R. Motz, *Dynamic Maintenance of an Integrated Schema*, PhD Thesis, Darmstadt, University of Technology, Germany, 2004.

[6] P. Fankhauser, *A Methodology for Knowledge-Based Schema Integration*. PHD-Thesis, Technical University of Vienna, December 1997.

[7] *Beneficios de una Arquitectura Orientada a Servicios*,
URL: http://www.developer.com/services/article.php/1041191 (August 2005).

[8] R. Cattell, *The Object Database Standard: ODMG–93*,
Morgan Kaufmann Publishers, ISBN 1-55960-396-4.

[9] E. Radeke, *Extending ODMG for federated database systems*.
Proc. 7th international Workshop on Database and Expert Systems Applications
(Setiembre 1996). DEXA. IEEE Computer Society, Washington, DC, 304

[10] F. Yergeau et al, *Extensive Markup Language 1.0*, W3C Recommendation, 3 ed., 2004.

[11] *XML DTD Tutorial*, URL: http://www.xmlfiles.com/dtd (August 2005)

[12] *Java 2 Platform, Enterprise Edition*, URL: http://java.sun.com/j2ee/ (August 2005)

[13] F. Alvarez, R. Amador, *Federador Link-All*, Proyecto de Grado, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, June 2005