Information integration among Heterogeneous and Autonomous Applications

Benabdelkader, A.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 6

GFI2S - Generic and Flexible Information Integration System

6.1 Introduction

Nowadays, for information management in complex organizations, a large variety of different database management systems (DBMSs) are used, which are at best chosen to meet the specific characteristics and requirements of every application environment. Existing application environments differ in their main characteristics and features. On one hand, they differ in their distributed/centralized architecture, their size, complexity, and the type of data they handle. On the other hand, their requirements depend on the global functionalities that they need to provide and on the required level of integration.

Following are brief descriptions of the main characteristics, which have a direct impact on the level of complexity of applications in terms of information management. These characteristics need to be considered when designing and developing information integration mechanisms for advanced application domains:

- **Type of Application**: due to their wide diversity of interest, emerging applications constitute a wide variety. Certain applications are specific to tasks that are internal to organizations, while some others are more dedicated to web environment and e-commerce. A third type focuses on building secure collaborative environments among a trusted community of users, and so on.

- **Size and Complexity of Applications**: existing organizations are of different complexities and may consist of a wide range in sizes. For example, organizations may be as small and simple as a local or centralized application where all modules can run on a single unit, or as large and complex as a heterogeneous application with many geographically distributed processing units.

- **Type of Data and Data Integration**: some applications are developed using simple data structures and may perform little information exchange with others. Other applications, however, handle inter-linked complex objects and require the integration of large data from external sources.

- **Level of Integration**: the required global services among heterogeneous and autonomous sites identifies the level of integration to be provided, while supporting the
heterogeneity of the underlying database components, and preserving their autonomy.

The scope and characteristics of application environments as described above, and their required level of interaction and integration with other applications, play the major roles in designing suitable database architectures and database management systems for them. In particular, for every application environment, the underlying DBMS must be carefully chosen to meet its specific requirements.

At the logical level, all existing DBMSs provide certain basic functionalities for application modeling, data structures, and the information storage and retrieval. However, at the physical level (the implementation level), DBMSs differ on the architecture that they rely on, the constructs they offer, the data storage mechanisms they use, and the information retrieval functions they provide. Therefore, existing DBMSs lack the possibility to be efficiently used for all types of applications. Some DBMSs better suit smaller applications, while others are more dedicated to complex environments and focus on the management of, for example, multimedia information and large data sets. Thus, *any attempt in the direction of forcing different applications to use the same database system for the management of all their information services is unrealistic*. Even within the same environment, in certain complex applications, the use of more than one DBMS cannot be avoided.

From the application cases, described in Chapters 3, 4, and 5, it is clear that a main criterion required by most advanced applications, is the provision of collaboration possibilities and data sharing/exchange among distributed, heterogeneous, and autonomous organizations. We also learned from the application cases that providing interoperability and integration among sites, via the deployment of database standards and emerging Internet technologies, is one of the most challenging approaches in the area of integrating heterogeneous information from autonomous sites. The interoperation/integration process eases the collaboration among existing systems, while preserving their autonomy and privacy in manipulating their own data independently of other database systems.

Research in the area of integrated information management systems started in the early 1980's. However, the integration focus since then has been stepwise, from supporting homogeneous centralized systems, to heterogeneous distributed systems, and finally towards the federated solutions. A number of these solutions are described in details in Chapter 2, where a comprehensive classification architecture for these approaches is also provided and illustrated. Considering the subject addressed by this chapter, we briefly summarize the main categories addressing the integration, as follow:

- **Tightly coupled homogeneous systems** denoting logically centralized, but physically distributed, databases systems [CP 84, SBD+83, AVF+92]. This approach lacks the support for data representation heterogeneity and does not preserve the autonomy of individual database systems participating in the collaboration network.

- **Tightly coupled heterogeneous database systems** where database schemas are integrated in one centralized global schema on top of which, database views can be defined [TBD+87]. This approach proves inefficient in terms of flexible information sharing and in preserving full autonomy of the local database systems.

- **Loosely Coupled federated architecture** proposed in [HM 85, LA 86], involves the integration of several database systems, stressing the autonomy and flexible sharing of information in a loosely coupled architecture. Examples of loosely coupled federated systems, providing partial autonomy to the involved databases in the network, are MRSDM [Lit 85], OMNIBASE [REM+89], and Calida [JPS+88]. To this architecture, Kim et al. [KCG+93] provides suitable techniques for resolving representational
conflicts within the context of multidatabase schema integration. Another example of federated information management approaches is proposed by the PEER federated system [AWH 94, ATW+94]. PEER follows the pure federation supporting information heterogeneity and preserving the full autonomy of nodes in the federation.

This chapter focuses on the design of a Generic and Flexible Information Integration System (GFI$_2$S). *Flexibility* in GFI$_2$S resides in its ability to add/remove new system to/from the federation with involvement of minimum effort. Flexibility of GFI$_2$S is supported through the use of the specific two-component architecture, while, its *Genericness* is achieved through the incorporation of database standards, emerging Internet technologies, and middleware solutions.

The design of GFI$_2$S is based on the investigation, evaluation, and validation of the methodologies and systems discussed within Chapter 2; and motivated by the expertise gained within the development of the various R&D projects addressed in Chapters 3, 4, and 5 of the dissertation. Thereby, the architecture of GFI$_2$S is inspired on, and extends the architectural design and development for these systems. A number of these aspects and design considerations are described in previous chapters, hereafter, we briefly summarize their main contribution to GFI$_2$S as follow:

- The Waternet system contributes to the design of GFI$_2$S at (1) the Node Federation Layer by tackling the fundamental schema management challenges, and at (2) the Local Adaptation Layer by serving the system openness through the adaptation and extension of the data adapter components.

- The MegaStore system contributes to the design of GFI$_2$S through (1) the deployment of database standards and Internet Middleware supporting system reusability, (2) the development of a parallel/distributed database server assuring system efficiency, and (3) the development of user friendly interfaces assisting advanced/ordinary users in accessing the underlying information sources.

- The VL information management framework contributes to the design of GFI$_2$S by addressing a number of important issues that are applied at every site to better enable it as a node within the federation. On one hand the data storage strategies and the performance issues assure the database efficiency at the local sites. On the other hand, the definition of restricted schemas and the development of universal and schema free tools to access them, allow sites to properly share part of their information.

6.1.1 Focus of GFI$_2$S

The architectural components of Generic and Flexible Information Integration System (GFI$_2$S) supports the integration of different types of data from heterogeneous applications, in order to achieve broader information access capability, and minimize development efforts. The architecture of GFI$_2$S is composed of two main components: (1) Local Adaptation Layer (LAL) that facilitates the access to the underlying databases in the node; and (2) Node Federation Layer (NFL) that provides links to the information and applications outside the node and supports the information sharing and interoperation. This two-component architecture of GFI$_2$S supports a wide variety of existing applications with efficient means for their interconnection and interoperation, while preserving their *heterogeneity*, *distribution*, and full *autonomy*:
• **Heterogeneity** refers to the fact that each database may apply its own distinct DBMS and data representation is heterogeneous in terms of structures and semantics.

• **Distribution** refers to the storage and processing of information from distributed data sources, located on different host computers.

• **Autonomy** refers to the fact that each site within the federation community is an independent system. Typically, a local site is pre-existing to the creation of a cooperation network and has its own administration policies and users groups.

The distinctive features of the GFI2S integration approach resides in: (a) the specific combination of database standards and Internet middleware with the fundamental research approaches, and (b) the way in which they are deployed and interlinked within the two specific components of the GFI2S architecture. These two considerations make the GFI2S approach distinct from all other existing federated/integrated approaches and introduce GFI2S as a generic solution providing a flexible architecture and an open facility for integration/interoperation among heterogeneous, distributed, and autonomous sites. Following are clarifications related to genericness, flexibility, and openness of GFI2S:

1. The use of object-oriented standards and middleware solutions, in the development of the GFI2S system, makes its architecture generic. Each site that wishes to join the federation only needs the knowledge about its “underlying database system” and about the “common format” adopted at the federation layer. The local users at each site gain proper expertise about the underlying local database characterization and specifications. At the same time, the common format adopted at the federation layer is widely understood by these users.

2. The use of a two-component architecture within GFI2S makes the system flexible. This flexibility is supported from two sides, on one hand, sites can easily join or quit the federation. On the other hand, the schema integration strategy followed at the node federation layer allows for a customized integration, which can be tailored to the need of each site. Within each site, the Local Adaptation Layer (LAL) facilitates the access to the underlying databases in the node, in their native format. While the Node Federation Layer (NFL) provides links to the information and applications outside the nodes, to support the information sharing and interoperating.

3. The binding of database standard mechanisms (e.g. ODMG standard) with emerging advanced tools in the domain of information management (e.g. Java, JDBC, and XML) makes GFI2S an open facility for sites integration. The deployment of advanced and standard tools facilitates the information integration and the interoperating among multi-platform and multi-language applications.

The main concepts related to the architecture of GFI2S system and the deployment of standard mechanisms and advanced tools are further described within the remaining sections of this chapter. Section 6.2 addresses the information integration approach of GFI2S. Generic and Flexible Integration Information System, and outlines its architecture. The architecture of GFI2S constitutes two components of Local Adaptation Layer (LAL) and Node Federation Layer (NFL). Further details of LAL and NFL layers of GFI2S are presented in sections 6.2.1 and 6.2.2. Within these two sections, detailed descriptions are also provided regarding the various components and concepts constituting both the local adaptation layer and the node federation layer. Among other components, section 6.2.2 also describes federated schema management, federated resources specification, federated query processing,
and the various schema definition and schema derivation primitives. Section 6.2.3 discusses the deployment of database standards and middleware solutions within GFI2S, to facilitate the collaboration among a large number of organizations. Section 6.2.4 describes an example operational case that shows the quality of GFI2S approach and illustrates the main benefits gained when deploying its architecture. Finally, section 6.3 concludes the chapter and emphasizes the major achievements of GFI2S in terms of information integration and systems interoperation.

6.2 GFI2S Information Integration Approach

This chapter addresses the design of a flexible approach for information exchange and interoperation among heterogeneous applications. The proposed approach follows the node-to-node federation described in section 2.2.2.2. Additionally, this approach extends the existing solutions via the deployment of standard solutions, middleware, and the emerging Internet and database technologies. Therefore, the information integration in GFI2S is addressed from two perspectives:

1. There will be a design and partial development of a generic and extensible integrated information management approach referred to as Generic and Flexible Information Integration System (GFI2S), and

2. It will involve the definition and extension of database standards and middleware solutions serving the interoperability requirements, and providing support for openness and flexibility to the GFI2S approach.

For federated schema management, the architecture of GFI2S has its roots in the PEER federated/distributed database system [TA 93, ATW+94], which was previously developed within the CO-IM group of the University of Amsterdam. Moreover, the designed solution takes advantages from other related research and approaches. Among the applied features we enumerate the use of database standards [BFN 94, PWD+99], object-oriented technologies [CBB+00], mediated systems [DD 99], wrappers [THH 99], Grid distributed technologies [ABB+01, BHG+01], and semantics resolution [KCG+93, SP 94, HM 99]. In the GFI2S system, the nodes store and manage their data independently, while in the case of common interest they can work together to form a cooperation network at various integration levels. This nesting of nodes at different integration levels allows for a variety of configurations, where, for instance, certain kinds of cooperation are formed to enhance the performance of data sharing, while others are formed to enhance the complex data sharing in a distributed or federated manner.

Figure 6.1 shows the high-level architecture of the GFI2S integration approach among several sites. The two main components participating in the design and development of the integration layer at each site within the federation community are the Local Adaptation Layer (called LAL) and the Node Federation Layer (called NFL). The local adaptation layer defines the set of concepts and mechanisms that: (1) facilitate the access to the underlying local data sources in their native format, and (2) control their access rights. The node federation layer acts as a mediated common interface that sits between the local system and other external data sources participating in the federation.

Previous work in the area of federated database systems has involved a considerable effort in integrating new systems to the federation community, where each federated system is considered to interface with all other native systems in the cooperation. Our approach
considers the use of standards at the federated layer and it is preferred that each site that wishes to join the federation defines and specifies the mapping to and from the federated layer to its internal data source. Thereby, all data source components in the federation only communicate using the standard languages and common data format, adopted at the federated layer, which are widely understood by a large community of users in the field.

Figure 6.2 illustrates the communication mechanism among two sites using GF\textsubscript{2}IS architecture, where the LAL overcomes the specificities of the underlying data source in terms of query languages and data format, and the NFL overcomes the data models specificities among several nodes within the federation community. More precisely:

- The LAL handles the communication between the standard data and query format and the specific data model and query language deployed at the underlying data source. It consists of a set of specifications and tools supporting the communication to the underlying data source using a common data model and a common query language.
- The NFL at each site: (1) defines the part of the information to be imported/exported from/to other sites within the federation community, (2) defines the mapping between import, export, and integrated schemas, and (3) specifies the circumstances under which external users can access exported information.
The GF12S architecture supports several features, which ease the integration process among autonomous and heterogeneous sites. These features are mostly achieved through specific design considerations:

1. The global federation is build based on the federated resources\(^1\) defined at the nodes federation layers (NFLs). The nodes federation layers are based on the use of standards for data modeling, for information sharing, and for query languages (e.g. ODL, XML, OQL, and SQL). These standard mechanisms are widely adopted in most applications, thus, facilitating the interoperation/integration process.

2. The Heterogeneity at every node is supported at the two levels of:
   - Database management systems, where different data modeling approaches are considered, ranging from relational to object-oriented and object-relational, and from file systems to legacy database systems.
   - Data representation, in which the same data may be defined differently (different schema definitions), in terms of its semantics and structures.

3. The proposed architecture makes it possible to add a new application to the federation, just as easy as to take one away. To join the federation, each application only needs knowledge about its underlying database system and about the common format adopted at the federated layer.

4. The user assistance in remote data sources integration and data access, is achieved via flexible interactions with the common data model at the federated layer, and through common query languages. The Universal Database Access and the Safe/reliable Data Export interfaces, described in Chapter 5, illustrate two examples of tools assisting GF12S users in integrating heterogeneous information sources and defining information visibility levels on them.

5. The use of standard mechanisms and middleware solutions for applications modeling, data structuring, and information retrieval ease the interoperation/collaboration process among a wide variety of preexisting applications. The database schema evolution and the multi-media data handling are supported via the deployment of object-oriented

\(^1\)The term resources in this context refers to information sources.
standards, while the emerging Internet technologies and middleware solutions allow universal access to the data, ease the information exchange mechanisms, and facilitate multi-platform applications development.

6. The access to the local data sources is supported via the specification of mappings between the local data sources and the common data model at the NFL, where the LAL, at each node, supports the controlled transfer of data to the NFL while preserving full local autonomy of that node. For the development of mappings of process also considers the use of object-oriented approaches that combine both elements of object-oriented research and heterogeneous distributed DBMS research (e.g. applying object-oriented concepts to heterogeneous and distributed database environment is considered).

7. To enforce the database integration process and facilitate the federation of a wide variety of heterogeneous information originating from distributed sources, the GFI₂S approach also defines the concepts of Semantics description and Dictionary of Terms. These two concepts, eventually provided by each participating database, help the conflict resolution about the meaning and similarity of objects in terms of their naming and structural representation. The importance of these components resides in providing a clear definition for all the objects exported by a database system, and facilitates their integration by other systems.

The remaining sections describe in details the design of GFI₂S and outline the features and considerations enumerated above, which are covered by the two components of GFI₂S, namely the Local Adaptation Layer and the Node Federation Layer.

6.2.1 Local Adaptation Layer (LAL)

In order to make the information sharing among heterogeneous and autonomous sites a reality, local sites must develop a set of mechanisms to facilitate access to their local data, to define the concept models, and to specify the circumstances under which external users/applications can gain access to a part of their information.

A good strategy for information integration must clearly distinguish between the tasks to be performed locally (specific individual tasks) and the common tasks that involve the contribution of several networked applications (common tasks of the federation). The design strategy we have followed considers that users at the federated layer only operate on common data models using standard mapping tools and querying languages, without caring about the specific operations for query translation and results transformation, which are left to each local site. Thus, it is preferable to keep the local mapping rules and derivation primitives, related to the query and to the result translations, local to each layer. For security and efficiency reasons, these specifications can be better supported as built-in components within the local adaptation layer. On one hand, performing these mapping locally at each site prohibits external users from knowing or needing to know the underlying data structures of the local application. On the other hand, the local execution of the queries and the transfer of the resulting data in a condensed format improve the interaction process and reduce the communication time between interoperable database systems.

The main goal of the Local Adaptation Layer, created at each local node, is two-fold: (1) to transform the local results of the queries into a format that is readily understood by other applications within the integrated system. Thus, the local adaptation layer at each node
provides a direct link between the local data source and the integrated system, forms the local database gateway, and implements the interfaces to the federated layer.

Figure 6.3 shows the main components of the local adaptation layer. Mainly: (1) the Local Interoperation Agent that forms the main gateway to outside for the local node; (2) the local resources specification that defines the access rights, the query mapping rules, and the results transformation mechanisms; and (3) the local query processor that interfaces to the local data source for query translation and execution. The dashed arrows from the local resources specification to the Local Interoperation Agent and to the local query processor, means that the operation processes of these latter components in terms of data access rights and query translation are performed and controlled based on the local resources specification. The next sub-sections further describe in more details the components of LAL.

![Diagram of Components of the Local Adaptation Layer](image)

### 6.2.1.1 Local Interoperation Agent (LIA)

The Local Interoperation Agent component acts as the main gateway to outside for the local node. Its main functionality include: checking and controlling the access rights to data, validating the query commands against the local resources specification, and coordinating the local query execution and result transformation. To properly accomplish the main tasks assigned to it, the Local Interoperation Agent operates according to the local resources specification defined at the Local Adaptation Layer of the corresponding node.

### 6.2.1.2 Local Resources Specification (LRS)

The Local Resources Specifications (LRS) help the Local Interoperation Agent and the local query processor with performing their tasks and controlling the access to the sharable data at the local node. In this repository, the local resources are specified in terms of the following features:

- **Access rights and user visibility levels**, which define the set of users that can access the data, and specifies the part of information to be shared with them.
• **Query mappings.** which define the set of rules for query mapping that help the local query processor (LQP) in translating the arriving queries from the common format into the local format used at each local database system.

• **Results transformation.** which define the set of rules for data translation that help the local query processor (LQP) in transforming the results of a local query into the common format. The common format is based on the use of standards, which are widely understood by the networked applications.

More details describing the resources specification and outlining the operational relationship between local and federated resources specification are given in section 6.2.2.2. The latter section outlines the schema definitions, mapping derivations, access rights definition, and semantics description related to the sharable information among distributed sites.

### 6.2.1.3 Local Query Processor

The *Local Query Processor* is a software component that offers a uniform querying interface, to the node federation layer. This querying facility applies to the schema defined on the underlying local data source. The role of the local query processor is to pose queries to the local data source and extract an answer resulting from that data source. The extracted result must be reorganized into the common format that can be understood and manipulated at the federated layer.

The local query processor, at each site's local adaptation layer, is usually created with two distinct parts, which consist of the *query translation* and the *result transformation*. The query translation component transforms a query in the common format into a query in the local querying language in order to be executed locally. The result transformation process transforms the local data resulting from the local query execution into a format that is readily understood at the integration layer.

**Local Query Translation & Execution:** At the local adaptation layer, before executing the submitted query, first the query must be translated from the common format into the local format of the underlying data source, and second the transformed query will be executed against the local data source. We assume that the federated query processor, as will be presented in details in section 6.2.2.4, only performs the decomposition of the federated query into a set of sub-queries that are still in the common format. The task of translating the sub-queries from the common format to each local format of the database systems participating in the federation is left to the local adaptation layer of each node. If we consider the integration approach introduced in Figure 6.1, the node making the integration does not need to know the specifications of local query languages at participating database, neither it needs to knows about the local DBMSs used by these databases. So it is easier and safer to process the relevant sub-queries translation at each local node than having the database integrator transforming all the queries from the common format to the various local formats.

**Local Result Transformation:** The local result transformation process translates the result of a sub-query into the common format defined at the federated layer. Similarly to the query translation process, since the integration layer follows a common model for data representation, the results of the local sub-queries are locally transformed at each node and sent back to the user/application that has requested it. The *Result Assembler* at the federated layer, as will be presented in details in section 6.2.2.4, will
merge this data and organize it to fit the federated schema specified by the requesting user/application.

Below, we summarize the steps involved within the execution process of a given query at the Local Adaptation Layer (LAL), which include:

1. A query is submitted to the local application through its LAL. The query is expressed using standard languages (e.g. OQL).
2. The Local Query Processor (LQP) translates the submitted query in order to be conforming to the local native schema definition and local query language. The query is then sent to the local source where the real data is stored.
3. Through the result assembler of LQP, the returned results (expressed according to the local format) are transformed and reorganized to fit the export schema defined at the node federation layer.
4. The global results, expressed using the standard format (e.g. XML, CDM), are returned to the user.

### 6.2.2 Node Federation Layer (NFL)

In recent years, database research is increasingly focused on the area of systems interoperability and information integration, where the research is approached differently through different methodologies such as the multidatabase systems (MDBS) [LMR 90], federated database systems [SL 90], or mediated systems [FLM 98]. There are however, a large number of challenges still need to be addressed and solutions to be found in this area. Therefore, further and deeper investigations are required in this field.

The Node Federation Layer of the integrated system, addressed in this section, describes the integration mechanism of GFI₂S. The global architecture of the node federation layer, presented in Figure 6.4, is composed of (1) a Federated Schema Management defined using the Object Modeling Language (ODL), (2) a set of Federated Resources Specification for the information sharing, and (3) a Federated Query processor based on standard query languages. The modeling language within the GFI₂S system allows the definitions of the set of entities, their attribute names, and relationships that are used to build the schema components. While, the common query language is the mean to formulate queries against those schemas.

![Figure 6.4: Node Federated Layer Representation](image)
In addition to supporting heterogeneity of the database components and preserving their autonomy, the NFL layer at each node allows every site within the federation to:

- Define as many export schemas as required: each export schema defines a part of the sharable information for certain external users, and is tailored to the specific need of those users.
- Only import the required parts of information from the schemas exported by other sites: related information is imported in coordination with the node federation layer of the exporting site.
- Design and build its own integrated schema: where classes and their attributes in the integrated schema represent the global overview of all the data that can be accessed by the site. The integrated schema represents the local information merged with all imported information.

The node federation layer is responsible for maintaining the local, export, and integrated schemas consistency, decomposing global queries into sub-queries executable by the local database at different sites, coordinating the execution of the sub-queries, and reorganizing the returned sub-query results to fit the global view of the integrated schema. These tasks are accomplished at the node federation layer based on the set of federated resources, which define the necessary specifications related to the sharable information, its derivation operations, and the circumstances under which external user can gain access to that information.

- **From the usage point of view**, the NFL at each node provides the means to create a seamless federated database, hiding details such as the database location, data representation, and heterogeneity from the users and application programs. Moreover, the integrated schema within the node federation layer is customized to the need of the users and to the requirements of their applications.

- **From the implementation point of view**, the aim of the NFL is to remove the need for static global schema integration and allow each application to have more control over the sharable information: control, is therefore decentralized. The node federation layer brings an open component-based architecture to build an integrated system with advanced database integration features, which makes it a favorable solution located in between the two extremes of no integration and total integration.

The Object Database Standard (ODMG 3.0) \cite{CBB+00} is chosen for specifying the modeling constructs and the querying language at the federation layer. In this direction, the object definition language (ODL) is used for describing the data structure and constraints, while the Object Query Language (OQL) is used for operating and retrieving the sharable information.

Besides following the standards in defining the database schema components, the ODMG model provides the following advantages:

- It is simple enough so that it can be readily understood and implemented.
- It is semantically expressive to capture the intended meaning of conceptual schemas that may reflect several kind of heterogeneity.
- It contains the basic features common to most semantic, hierarchical, relational, and object oriented models. It supports modeling primitives, type membership, object properties, and object behavior.
• It includes the ability to encapsulate the functionality of shared objects, its extensible nature, and object uniformity.

• It is an open standard for new extensions regarding languages programming, Java binding, and interfacing with other emerging standards and technologies.

The use of a common data model, that is object-oriented, does not rule out the participation of other relational, hierarchical, or file system based models. Rather, it assures their full integration via the support of their semantics heterogeneity through the expressive object data model adopted at the federated layer.

The remaining of this section will address in more details different components that constitute the federated layer at each node participating in the federation community. Namely, the following sub-sections describe the steps required for defining:

1. The **Federated Schema Management**, which provides the necessary mechanisms for the definition and derivation of the local, export, and integrated schemas (described in section 6.2.2.1).

2. The **Federated Resources Specification (FRS)**, which defines the set of concepts related to the sharable information, to their derivation, and to their access right policies (described in section 6.2.2.2).

3. The **Federated Derivation Primitives**, which specifies the mapping rules for schemas derivation and query decomposition (described in section 6.2.2.3), and

4. The **Federated Query Processor (FQP)**, which processes queries that may require extracting and merging data from multiple sources (described in section 6.2.2.4).

### 6.2.2.1 Federated Schema Management

The federated schema management is based on the integration of information from several remote and local data sources. As depicted in Figure 6.5. Every site participating in the integrated system is represented at its node federation layer by several schemas:

• The **Local Schema**, which models the data stored locally. This schema represents the part of data owned by the node.

• The **Export Schemas**, which model the part of information that this node wishes to make accessible to other nodes/users.

• The **Imported Information**, which represents the part of information that this node wishes to import from other exported schemas participating in the federation.

• The **Integrated Schema**, which merges local and imported information into a coherent view that satisfies the node's requirements in term of information management.

The schemas representation adopted for the node federation layer, illustrated in Figure 6.5, guarantees the following features:

• **Autonomy is preserved**: each node has full control on its local data. decides on the part of data to be shared with external nodes, and only imports the part of information needed from the various schemas exported by other nodes.
As many export schemas as needed: each node individually decides on the part of data that will be shared with the external nodes based on other node's privileges and rights.

Each node decides on the layout and the format of the data that they want to export and make available to the outside world.

The information managed by the integrated system is maintained and stored at several local and remote heterogeneous data sources where it is generated or it belongs.

The following sections describe in more details the components of the Federated Schema Management that support the definition of local, export, and integrated schemas; while their corresponding resources specification, semantics description, and derivation primitives are further described in sections 6.2.2.2 and 6.2.2.3.

Local Schema

The local uniform schema at the NFL reflects the data structure of the underlying local data source, and it is defined using a common-format adopted at the level of the federation layer to represent all the local schemas at the sites. Under normal situations, this schema can be semi-automatically constructed from the native schema of the underlying DBMS, with the help of a local user.

The local uniform schema supports the derivation of a set of export schemas (serving the requirements of other applications) for information sharing. The local uniform schema defines a consistent data model facilitating the manipulation of the shared information.

For the derivation of export schemas, another strategy that can be adopted is to define them directly on the local underlying data source, without passing through the definition of the “local uniform schema”. However, the usage of the local uniform schema as an intermediate step, eases the definition and further expansion of the export schemas at different phases of the integration process, as well as making the federation more flexible when new information sharing policies are required.
The native schemas are then transformed using the standard data modeling constructs adopted at the federation layer. Thus, the problems related to the data modeling heterogeneity among applications are solved, while representational heterogeneities are also treated at the node federation layer, for which related concepts to schemas derivation and semantics description are presented in sections 6.2.2.2.

**Export Schema**

The export schema defines a part of local information that a node desires to share with the outside world. It also specifies the “mapping rules” for translating the data from the local data source to the various export schemas. These mappings are necessary when a node decides to export data in a different format than the local representation. Such a variation is necessary for security reasons and due to some required layout when data needs to be exported. A node for instance may create a single export schema for each user/application or it may define a common export schema that can be exported to different nodes.

A set of resources are defined within each export schema, these resources specify the derivation mappings from the local schema to the export schemas. define their access rights, and provide the corresponding semantics descriptions, which help during the integration process. These resources are further described in section 6.2.2.2.

**Integrated Schema**

Within each node’s federation layer, the Integrated Schema (INT) represents a global and coherent overview of all accessible local and remote information. Its definition is based on both local and imported information; that are merged in a way that makes the physical/logical distribution of information transparent to the user. The integrated schema can be interpreted as one user’s global classification of objects that are organized differently by the schemas in other data sources.

At the NFL of every node, local and remote information from different sources are modeled into one data model, representing the integrated schema\(^2\). The constitution of the integrated schema of a node is based on the need of the application and on the part of information that this application is authorized to access and retrieve from other sites. Therefore, the integrated schema represents the structure of the local and imported sharable information. The use of a common data modeling language in defining the integrated schema plays a vital role in resolving some aspects of the data model heterogeneity and eases both the schema integration task and the mapping specifications.

The integrated schema for each system incorporates sub-schemas of the local data sources. The Local Adaptation Layer (LAL), developed at each local data source, provides the link that transforms the local database instances into instances of the integrated schema for the node. Consequently, queries on the integrated schema can access the data that resides in remote data sources of the network.

If the local schema uses a different data modeling constructs and a different query language than the languages used within the federation layer, the local schema is seen as an export schema for which, again the mapping specification and the development of the Local Adaptation Layer become necessary. Thus, the import of information from the local schema to the integrated schema will follow the same process as if importing it from a remote node.

\(^2\)The schema integration component also considers both schema and database browsing, schema modification and enrichment, and the interactive, incremental construction of integrated schemas for use by different agents within the cooperation community.
The diagram in Figure 6.6 provides an example representing the schema integration and systems interoperability within a collaborative environment among three departments of Manufacturing, Sales, and Customer management. Within this environment, users commands are by default, formulated and evaluated against the integrated schema. However, it is always possible, for instance, for database administrator users to specify and submit queries against a local, or export schema as long as they hold proper access rights.

The exchange of information between integrated systems within a federation community is done through messages, where a message content is of two types query or data. Queries are always sent from the integrated database system to the local and remote information sources, while data is passed from the remote nodes to the integrated system as an answer to its query. The content of a message, query or data, is expressed using respectively the common query language (e.g. SQL and OQL) and the common data format for information exchange (e.g. XML and OIF).

Within the integrated schema, a different data representation can be used. Similarly to the case of data export in which an application decides on the layout for representing the information to be shared with the outside world, a node at the integrated system can also decide on the way to reorganize or merge the imported data with the local information. Therefore, there is also a need to specify a set of resources that serve the federation process. These resources, referred to as Federated Resources Specification (FRS), define the necessary concepts for deriving the integrated schema and specifying the access rights to it.

### 6.2.2.2 Federated Resources Specification

In order to facilitate and control the access to their data, each node specifies a set of resources, serving the federation process. These resources define the set of schemas and concepts related to the description of the sharable information and also specify the circumstances under which a node can join and register within the federation community. The main distinction between the federated resources specification and the local resources specification (presented in section 6.2.1.2) is that: the local resources specifications define the mapping rules and the access rights to the local underlying data source, while the federated resources specify the concepts related to the integration through the Node Federation Layer.
Within each node participating in the collaboration community, The federated resources include definitions and specifications of the following five components:

1. **Export schemas definition and derivation**, respectively using the object definition language and the schema derivation language. An export schema definition is identified by an \'.odl\' extension (\'<export schema>.odl\'), while its derivation is identified by a \'.drv\' extension (\'<export schema>.drv\').

2. **Export schemas registration**, in order to enable other sites to access and share its data, each site must register the set of export schemas together with their access rights specifications.

3. **Integrated schema definition and derivation**, similarly to the export schema definition and derivation, the integrated schema is defined by two components, its definition and its derivation (e.g. \'<integrated schema>.odl\' and \'<integrated schema>.drv\').

4. **Access rights definition for export and integrated schema components**, which are defined by the extension \'.acs\', namely, \'<export schema>.acs\' and \'<integrated schema>.acs\'.

5. **Semantics description for export and integrated schemas**, which consist of a dictionary of terms and a dictionary of semantics, respectively expressed by \'<export schema>.dic\' and \'<integrated schema>.dic\'.

### Export Schemas Definition and derivation

Export schema definitions constitute subsets of the local schema definition. To define the export schema, we preferably use the ODL object definition language. Figure 6.7 shows a simplified diagram of a basic example for the export schema definition. In this example, a new class *Person* in export schema *EXP2* is derived from the class *Employee* in local schema *Loc*. The structure and the attribute names in *EXP2* are defined differently than in the local schema *Loc*.

![Figure 6.7: Export Schema (Exp2) Definition and Derivation - Example](image)

Figure 6.8 shows example derivation mappings for the class *Person* within the export schema *Exp2*, derived from the local schema *Loc*. The derivation process is based on a set of rules, which define the relationship between the class and attributes in the derived schema (*Exp2*) and those in the base schema (*Loc*). The example below shows the use of three types of derivation: rename, user-defined functions, and type casting.

- The class *Employee* and its attribute *LastName* are renamed in *Exp2* respectively as *Person* and *Name*.
- Attributes *Initial*, *Address*, and *Age* are derived using user-defined functions (respectively *GetInitial*, *Concatenate*, and *DateToAge*). The concept of user-defined function
(that will be later addressed in section 6.2.2.3) is simple and powerful: it allows a user to define his/her specific derivation's functions and algorithms.

- The attribute *Weight* is derived through type casting by converting the *float* value of *Weight* in the base schema into a value of type *short* in the derived schema.

```plaintext
// Derivation specification of class Person in export schema Exp2
Person@Exp2 = Employee@Loc
Person.Name@Exp2 = Person.LastName@Loc
Person.Initial@Exp2 = GetInitial(Employee.FirstName@Loc)
Person.Address@Exp2 = Concatenate(Employee.Street@Loc, Employee.City@Loc)
Person.Age@Exp2 = DateToAge(Employee.BirthDate@Loc)
Person.Weight@Exp2 = (Short) (Employee.Weight@Loc)
```

Figure 6.8: Export Schema (Exp2) Derivation - an example

The example described above shows the flexibility provided by our approach for users to define the derivation mapping from the base schemas. More details about the different derivation mapping concepts and their possible implementation strategies will be provided in section 6.2.2.3.

**Export Schemas Registration**

Database administrators at each local site must identify and register the set of export schemas that can be accessed by other sites and external users. The registration process of an export schema identifies its name and its location, along with other information needed for user access.

The following BNF-like notation defines the syntax for export schema registration:

```
Registration  ::=  <Export Schema> [description>] <host> <port> <mode>
Export Schema  ::=  identifier
port  ::=  number
Mode  ::=  R | W | R/W
```

The example below shows the registration process defined for two exports schemas Dublin Core (DC) and Traffic System (TS):

```
DC  Dublin Core  www.science.uva.nl  8800  R/W
TS  Traffic System  146.50.1.188  8900  R
```

The export schema registration allows the database integrators at other sites to identify the set of export schemas that can participate in the definition of their integrated schemas.

**Integrated Schema definition and Derivation**

The integrated schema defines the complete set of information that a site can access and retrieve. Therefore, this schema is derived from the local schema and from various schemas imported from other nodes within the federation network. Figure 6.9 for instance, shows a simple example of the integrated schema (INT) definition. In this example, a new class *Organization* in integrated schema INT is derived from the existing classes *Department* in schema Exp7 and *Faculty* in schema Exp3. This example involves the definition of new class *Organization*, which contains information from two other classes *Department* and *Faculty* that belong to different schemas Exp7 and Exp3.
Figure 6.9: Integrated Schema Definition and Derivation - Example

Figure 6.10 shows the derivation mappings of the integrated schema \((INT)\). This is based on a set of rules defining the relationship between the classes and attributes in the derived schema \((INT)\) and those in the base schemas \((Exp7)\) and \((Exp3)\). This example primarily shows the use of the \textit{Union} mapping and a user-defined function \((\text{Sum})\): both operations imply the combination of data originating from different schemas. The derivation mapping is reached using an extended language for schema derivation that supports the derivation operations (e.g., rename, union, subtract, and user-defined functions). More information about different derivation mapping languages and primitives will be provided in section 6.2.2.3.

```
//--- Derivation Specification of class Organization in integrated schema INT
Organization@INT = Union (Derive(Department@Exp7,
    Name=DeptName,
    NumOfEmployees=Employees,
    Interest=Field),
    Derive(Faculty@Exp3,
    Name=Faculty,
    NumOfEmployees=Sum*(Staff, Researchers),
    Interest=Area))
  * int Function Sum (int n, int m) return (n + m);
```

Figure 6.10: Integrated Schema Derivation – an example

Access Rights Specification for Users and Applications

This process limits the user accesses to the export/integrated schema components and identifies the set of allowable operations submitted on the sharable information. The access to data will be limited to the set of authorized users/applications. This process acts as a semantic integrity checker and access controller sitting between export/integrated schemas and the components accessing them. The access specifications are defined for each export and integrated schema. The following BNF-like notation is adopted for their specifications:

Access ::= <user> <password> | <user> <host> <port> <mode>
User ::= identifier
Port ::= number
Mode ::= R | W | R/W
The example below shows the access rights for three different users, defined for a given export or integrated schema:

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Username</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>R/W</td>
<td>8800</td>
</tr>
<tr>
<td>Toto</td>
<td>R</td>
<td><a href="http://www.science.uva.nl">www.science.uva.nl</a></td>
</tr>
<tr>
<td>Toto</td>
<td>W</td>
<td>********</td>
</tr>
</tbody>
</table>

**Semantics Description**

Structural characteristics and assigned names within the database schema do not sufficiently describe their real-world meanings among a number of interoperable systems. In order to support sharing of information among a collection of heterogeneous and autonomous sites, we must overcome the heterogeneity of type definitions among these databases. Thus, it is necessary to have an explicit dictionary that clearly defines entities of the database model using a natural language to describe the meaning of names used for types and their attributes. Availability of such a dictionary facilitates the integration of new export schemas from other sites, without the need for on-line support from external users, submitting those schemas. The semantics description also helps in developing *Syntax Match Assistance* tools, which facilitates the automatic/semi-automatic resolution of semantic and representational differences that occur among local and imported related data objects in different systems. Hammer and McLeod [HM 99] distinguish between two aspects in resolving those representational differences: (1) determine the relationships between sharable objects in different components, and (2) detect possible conflicts in their structural representations. The development of such a facility within the federation layer assists and helps in applying methods for resolving representational differences among a number of databases participating in the federation.

Existing approaches for conflict resolution in integrated database systems are based on heuristics [HR 90, NEM+86, SLC+88], classification [SSG+91], semantic proximity concept [SK 93], or fuzzy and incomplete knowledge [FN 92, VH 93], etc. These approaches may lead to inaccurate resolution of the semantic problems. Our approach considers the enforcement of structural representation and semantics resolution by the definition of a *Dictionary of Terms* and a *Dictionary of Semantics* both provided for each database schema at the Node Federation Layer. The *Dictionary of Terms* serves for automatic conflict resolution about the meaning and resemblance of objects in terms of their naming and structural representation, while the *Dictionary of Semantics* helps the schema integration process and to manually solve conflicts that are not automatically detected. The database schema integrator also uses the dictionary of semantics in order to understand the structure of each database schema for the participating sites in the federation.

For the integrated schema, one global dictionary is required per application environment to describe the terms and the semantics (*Terms.dic, Semantics.dic*). For the export schemas, also one dictionary is required for each export schema (e.g., *<Exp>Terms.dic* and *<Exp>Semantics.dic*).

---

3Heterogeneity in names and data structures is a natural consequence of independent creation and evolution of autonomous databases that are tailored to the specific requirements and characteristics of each application.
Hereafter, are the BNF-Like descriptions of the dictionary of terms and the dictionary of semantics, followed by some examples:

**Dictionary of Terms**

Term ::= <identifier><relationship><identifier>,[]
Identifier ::= literal
Relationship ::= X-equal | M-equal | Synonym
X-equal ::= syntactically equal
M-equal ::= semantically equal

*Example of Terms:*

'Hardware' M-equal 'Device'
'Device' Synonym 'Machine'
'Serial number' M-equal 'Code'

**Dictionary of Semantics**

Semantic Concept ::= <class name>:[<class description>]
                    [<attribute definition>]
class name ::= identifier
class description ::= literal expression
attribute definition ::= <attribute name>:<attribute description>
attribute name ::= identifier
attribute description ::= literal expression

*Example Semantics:*

Hardware: this class can be a Device or a machine
          h_name: hardware name
          h_SN: hardware serial number

6.2.2.3 Federated Derivation Primitives

Research work in the direction of defining a set of derivation primitives still lack standards and comprehensive necessary concepts for schema derivation and for mapping specification. Radeke [Rad 96] proposes an ODMG extension for federated database systems, in which the schema derivation is supported through the filtering process, based on the DROP command. Within the exported schema the DROP command specifies the attributes, relationships, and methods to be filtered from the local schema. However, the approach does not define a clear export schema for the shared data, rather it gives the list of concepts to be filtered from the local schema. Therefore, the complete elements of the local schemas at the underlying database are not hidden from the user. While, the main aim in collaborative systems is to limit the external access to only the part of information to which users gain access rights. Similarly, Busse et al. [BFN 94] describes an approach to introduce the notion of virtual classes for the evolving object database standard ODMG. However, virtual classes are represented as views, for which the derivation and mapping information is given as a queries that provide full instantiation of the virtual classes. Since the approach assumes a one-to-one correspondence between the integrated instances and due to the complexity in
integrating the relationships, the approach does not properly adapt to distributed databases and support for their different query languages and data models is not fully supported. Recent work presented by Roantree et al. [RKB 01] provides an extended object definition language for view schema integration, based on the standard model for object-oriented databases. It also provides ODLv language for view specifications and ODLw language for wrappers specifications. In our approach, we consider the schema integration to be addressed from two perspectives: (1) providing means for export and integrated schemas definitions, and (2) defining the derivation operations that maps the classes and attributes in the base schemas to those in the derived schemas.

Regarding point (1), for export and integrated schemas definition. we use an object definition language specifications to define these schemas. The ODL syntax is used to define the classes of the derived schemas and to specify the relationships between these classes. Thereby. an export schema, for instance. differs from other schemas by only being a subset of an existing local schema, while an integrated schema is a subset of the union of a number of local and export schemas of other sites.

Regarding point (2), related to derivation operations. a strong language for schema derivation must be designed and developed. The schema derivation language must define the derivation mapping for the exported/integrated classes and their attributes. The approach we propose for the derivation operations, distinguish between two types of derivation: class derivation and attribute derivation. These derivation primitives define the mapping between the derived classes/attributes in the derived schema and the basic classes/attributes in the base schemas. The schema derivation language suggested here is based on the PEER Schema Definition and Derivation Language SDDL [AWT*94, WA 94], and extends their definitions in order to achieve an open strategy for the schema derivation. Our approach also considers derivations based on the user-defined functions, especially, in the case of the attributes instantiation. For the class derivation the following constructs are supported: rename, union, restrict, and subtract. The attribute derivation can use any user-defined function to derive new instances from the base schemas. The rename construct is also supported for the attribute derivations.

Two approaches are suggested for the efficient implementation of the class and attribute derivation. In the first approach, the various functions for the class and attribute derivation will be provided in form of shared libraries or dynamic link libraries (DLLs), which will be linked to the node federation layer (NFL) of each site. A second possibility, is to define these functions as persistent stored modules (SQL/PSM) or Call-Level-Interfaces (SQL/CLI) within the SQL environment. Shared libraries will be provided by the developers for interfacing through the programming languages within C, C++, Pascal, Delphi, etc. while, SQL/PSM and SQL/CLI are provided for interfacing through standard database connectivity mechanisms such as OBDC, JDBC, and Embedded-SQL.

The following sections describe in more details the mapping operations/constructs for attribute and class derivation. In the given descriptions: $C$ stands for `\texttt{class-name@schema-name}' and $A_C$ represents the set of all attributes in class $C$. The domain of $C$ is denoted by $\text{Dom}(C)$ and the domain of $A_C$ is denoted by $\text{Dom}(A_C)$.

\begin{align*}
\mathbf{C} &= \{ \text{all legal classes of schema } S \} \\
A_C &= \{ \text{all legal attributes in class } C \} \\
\mathbf{A}_C &= \{ \text{all legal attributes of schema } S \} \\
\text{Let } I : \mathbf{C} &\rightarrow \text{Dom}(\mathbf{C}) \\
I(C) &= \{ \text{the set of all instances of class } C \}
\end{align*}
6.2. GFI₂S Information Integration Approach

\[ I(C) \subseteq Dom(C) \text{, where} \]
\[ Dom(C) = \prod_{a \in A_C} Dom(a) \]

**Attribute Derivation**

A derived attribute is defined by an attribute derivation expression as follow:

\[
\text{derived-attribute-definition} := \text{derived-attribute-name} = <a-expr>
\]
\[
a-expr := \text{class-name.attribute-name[@schema-name]} \mid \text{user-defined-function}
\]

Attribute derivation is accomplished by specifying an attribute either as a *rename* of another attribute or by a *user-defined function*. The semantics of the primitives defined for the attribute derivation are provided below, where \( a \) represents an attribute in class \( C \) and \( Dom(a) \) represents all the legal values of attribute \( a \).

\[ Dom(A_C) = \bigcup_{a \in A_C} Dom(a) \]

The two constructs, defined for the attribute derivation, are defined below.

**1- Rename**

The attribute *Rename* operation derives a new attribute \( a \) from the attribute \( a_1 \), where \( a \) and \( a_1 \) are two attributes in classes \( C \) and \( C_1 \).

\[
C.a = C_1.a_1
\]  
(6.1)

Represents:
\[
\forall a_1 \in I(C_1), \exists o \in I(C) : o.a = o_1.a_1
\]

Where:
- \( o, o_1 \) are two objects (instances) of classes \( C \) and \( C_1 \) respectively, and
- \( a, a_1 \) are attributes of classes \( C \) and \( C_1 \) respectively.

*Example*:

\[
\text{Organization.Name} = \text{Department.DeptName@Exp7}
\]

The attribute *Name* in the class *Organization* is derived from the attribute *DeptName* of class *Department*.

**2- User Defined Function**

The *User-defined-function* derives a new attribute \( a \) from a set of attributes \( a_1, a_2...a_n \) by applying the function \( f \), thus.

\[
a = f(a_1, a_2, \ldots, a_n)
\]  
(6.2)

Represents:
\[
f : Dom(a_1) \times Dom(a_2) \times \cdots \times Dom(a_n) \to Dom(a)
\]
Where $a$ is the derived attribute by applying the function $f$ to the attributes $a_i$ in the base classes $C_j$, $1 \leq j \leq n$.

Following is a simplified BNF-Like notation for the user-defined function concept:

$\text{user function} :: <\text{type}> <\text{identifier}> ([\text{param-list}]) <\text{function-body}>
\text{param-list} :: <\text{type}> <\text{identifier-list}>,
\text{function-body} :: <\text{statement-list}>
\text{statement-list} :: a \text{ list of statements in a certain programming language}
\text{identifier-list} :: a \text{ list of identifiers}
\text{identifier} :: an \text{ identifier in a certain programming language}

\text{Example:}
\text{Organization.NumOfEmployees@INT} = \text{Sum} (\text{Faculty.Staff@Exp3}, \text{Faculty.Researchers@Exp3})

The attribute \text{NumOfEmployees} of class \text{Organization} in the integrated schema (INT) is derived as summation of the attributes \text{Staff} and \text{Researchers} in export schema (Exp3) from the class \text{Faculty}, where \text{Sum} is a user-defined function with two parameters of type \text{Integer}. The \text{Sum} function can be defined as follow:

\text{int Sum (int n1, int n2) \{return n1 + n2;\}}

Class Derivation

In schema derivation, a derived class is constructed from other classes using \text{Union}, \text{Restrict}, \text{Subtract}, and other primitives as described below. A derived class is defined by a class derivation expression as follow:

derived-class-definition ::= derived-class-name = <c-expr>
c-expr ::= class-name@schema-name |
union (<c-expr>, <c-list>) | Subtract (<c-expr>, <c-expr>) | Restrict (<c-expr>, restriction) | Derive (<c-expr>, <derived-attribute-definition>)
c-list ::= <c-expr> | <c-expr>, <c-list>

The following constructs provide the semantics for the class derivation.

\textbf{1- Rename}

The class \text{Rename} operation derives a new class $C$ from another class $C_1$.

\[ C = C_1 \quad (6.3) \]

Represents:

$I(C) = I(C_1)$, where $C$ is the derived class and $C_1$ is a class in the base schema $S_1$.

\text{Example:}
\text{Organization@INT} = \text{Faculty@Exp3}

The class \text{Organization} in the integrated schema INT is derived from the class \text{Faculty} in export schema Exp3.
2- Subtract

The class *Subtract* operation derives a new class $C$ by subtracting instances of class $C_2$ from class $C_1$.

\[ C = \text{subtract}(C_1, C_2) \]  
(6.4)

Represents:
\[ I(C) = I(C_1) \setminus I(C_2), \]
where $C$ is the derived class and $C_1$ and $C_2$ are classes in base schemas $S_1$ and $S_2$.

*Example:*

\[ \text{Organization@INT} = \text{subtract} \ (\text{department@Loc, Faculty@Exp3}) \]

3- Restrict

The class *Restrict* operation derives a new class $C$ from a class $C_1$ based on certain restriction predicate $P$.

First, let us define a comparison predicate $\text{Comp}$ as follow:

\[ \text{Comp} = \{A \theta V\} \]

- $A$ is an attribute
- $V \in \text{Dom}(A)$
- $\theta \in \{\leq, \geq, =, <>, <, >\}$

We add the *Not* operator to the $\text{Comp}$ predicate:

\[ \text{Comp} = \text{Comp} \cup \{\text{Not}(C) \mid C \in \text{Comp}\} \]

Now we define a global predicate:

\[ P = P_1 \oplus P_2 \]

- $P_1, P_2 \in P$ OR $P_1, P_2 \in \text{Comp}$
- $o \in \{\text{AND}, \text{OR}\}$
- $C_1 \in C, P \in P$

\[ C = \text{restrict}(C_1, P) \]  
(6.5)

Represents:
\[ I(C) = \{o \in I(C_1) : P(o)\}, \]
where $C$ is the derived class and $C_1$ is a class in a base schema $S_1$.

*Example:*

\[ \text{Organization@INT} = \text{restrict} \ (\text{department@Exp7, Employees<25}) \]

Following is a BNF-like description of restrictions:

- \text{restriction} :: <predicate>[ [ <\text{AND}|\text{OR}|\text{NOT} > <predicate>, ]

- \text{predicate} :: <attribute><operator><range-attribute>

- \text{operator} :: = | < > | >= | <= | < | >

4- Union

The class *Union* operation derives a new class $C$ from a set of classes $C_1, C_2, \ldots, C_n$.

\[ C = \text{Union}(C_1, C_2, \ldots, C_n) \]  
(6.6)
Represents:
\[ I(C) = (I(C_1) \cup I(C_2) \cup \ldots \cup I(C_n)), \]
where \( C \) is the derived class and \( C_i \) are classes in a certain other base schemas \( S_j, j \in [1..k] \).

Example:

\[ \text{Organization@INT} = \text{Union} (\text{Department@Exp7}, \text{Faculty@Exp3}) \]

The class \textit{Union} operation will be demonstrated in more details in the next example with the \textit{Derive} construct. The \textit{Derive} operation Derives a new class \( C \) from the a class \( C_1 \) by applying the attribute derivation operations.

5- Derive

The class \textit{Derive} operation derives a new class \( C \) from another class \( C_1 \) by applying a number of expressions \( E_i \) specifying the attributes derivation.

\[ C = \text{Derive}(C_1, E_1, E_2, \ldots, E_k) \quad (6.7) \]

\( E_i \) is an expression defining the attributes derivation. It applies the attribute derivation primitives, namely, attribute rename and user-defined function:

- \( E_i : a_i = a_j \), where \( a_i \) is an attribute of class \( C \) and \( a_j \) is an attribute of class \( C_j \).
- \( E_i : a_i = f_i(a_1, a_2, \ldots, a_j) \), where \( a_i \) is an attribute of class \( C \) and \( a_1, \ldots, a_j \) are attributes of classes \( C_1, C_2, \ldots, C_j \).

\[ \text{Attributes}(C) = \{\text{range}(f_i)\}_{i \in [1..k]} \]

\[ I(C) = \{V \in \text{Dom}(C) \mid \exists o_1 \in I(C_1) : V = (E_1(o_1)\ldots, E_k(o_1))\} \]

or

\[ I(C) = \{V \in \text{Dom}(C) \mid \exists o_1 \in I(C_1) : V = (a_1 = E_1(o_1)\ldots, a_k = E_k(o_1))\} \]

Figure 6.11 illustrates a derivation example of a simple integrated schema \( INT \). First, we define three simple schemas (\( Exp/7, Exp/3 \), and \( INT \)), using the ODL specification. Second, we specify the derivation operations of the class \textit{Organization} in the integrated schema \( INT \). In this example, the derivation is specified in two steps:

1. Specify the derive operations at each of the classes \textit{Department} in \( Exp/7 \) and \textit{Faculty} in \( Exp/3 \). The derive operations specify the expressions related to the derivation of the corresponding attributes of class \textit{Organization}.

2. Derive the class \textit{Organization} in the integrated schema \( INT \) as a \textit{Union} of the derived classes \textit{Department} and \textit{Faculty} from the export schema \( Exp/7 \) and \( Exp/3 \).
6.2. GFI₂S Information Integration Approach

/// --- Definition of Schema Exp7
Class Department : Persistent {
    attribute String DeptName;
    attribute Short Employees;
    attribute String Field;
}

/// --- Definition of Schema Exp3
Class Faculty : Persistent {
    attribute String Faculty;
    attribute Short Staff;
    attribute Short Researchers;
    attribute String Area;
}

/// --- Definition of Schema INT
Class Organization : Persistent {
    attribute String Name;
    attribute Short NumOfEmployees;
    attribute String Interest;
}

/// --- Derivation Specification of Schema INT
Organization@INT = Union (Derive(Department@Exp7,
    Name=DeptName, NumOfEmployees=Employees, Interest=Field),
    Derive(Faculty@Exp3,
    Name=Faculty, NumOfEmployees=Sum*(Staff, Researchers), Interest=Area))

* Int function Sum(int s, int t) return (s + t)

Figure 6.11: Schema Definition and Derivation Specification - an example

Figure 6.12 illustrates an instantiation example corresponding to the derivation primitives, defined in figure 6.11. The example also shows the cases of attribute rename and user-defined function (e.g. Sum).

<table>
<thead>
<tr>
<th>DeptName</th>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Science</td>
<td>22</td>
</tr>
<tr>
<td>Physics</td>
<td>37</td>
</tr>
<tr>
<td>Mathematics</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
</tr>
<tr>
<td>Experimentation</td>
</tr>
<tr>
<td>Education</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Faculty</th>
<th>Staff</th>
<th>Researchers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Science</td>
<td>33</td>
<td>15</td>
</tr>
<tr>
<td>Medicine</td>
<td>45</td>
<td>37</td>
</tr>
<tr>
<td>Sociology</td>
<td>22</td>
<td>23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
</tr>
</tbody>
</table>

// Class C

<table>
<thead>
<tr>
<th>Name</th>
<th>NumOfEmployees</th>
<th>Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Science</td>
<td>22</td>
<td>Research</td>
</tr>
<tr>
<td>Physics</td>
<td>37</td>
<td>Experimentation</td>
</tr>
<tr>
<td>Mathematics</td>
<td>12</td>
<td>Education</td>
</tr>
<tr>
<td>Science</td>
<td>48</td>
<td>Research</td>
</tr>
<tr>
<td>Medicine</td>
<td>82</td>
<td>Unknown</td>
</tr>
<tr>
<td>Sociology</td>
<td>45</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

Figure 6.12: Classes and Attributes Instantiation - an example
6.2.2.4 Federated Query Processing

The task of the Federated Query Processor (FQP) is to process and respond to queries that may require accessing authorized data, and extracting and combining data from multiple sources. The sources are usually heterogeneous and geographically distributed. In other words, the federated query processor performs the functions of coordinating the sub-queries execution and provides a gateway to the multiple distributed data sources, where this complexity and distribution are hidden from the user; namely they are indistinguishable to the user from that of accessing a single database.

Figure 6.13 illustrates the steps involved in a federated query execution, which include:

1. The federated query processor first decomposes the query posed against the integrated schema into sub-queries on the local/remote data sources with their own local schema.
2. Various sub-queries are executed locally at the corresponding nodes and the sub-results are sent back to the federated query processor.
3. The federated query processor combines the local partial results into the common format and sends the complete results to the user requesting it.

The local and federated query processors are assisted in their functionalities with the set of resources specified at the local and federated layers.

![Figure 6.13: Federated Query Processor - The Steps](image)

After defining the federated layer for different data sources within the federation, authorized users can either access the data in a tightly coupled way by creating their integrated systems or in a loosely coupled manner via direct on-line access to the shared data. Therefore, queries can also be individually submitted to an export schema at each node's federated layer without passing through the integrated schema. When a query is formulated against the export schema, the following actions will take place:

- The query will be rewritten by the Local Adaptation Layer (LAL), in order to be conforming to the local query language. The query is translated based on the local resources specifications (LRS) of the underlying data source,
• The query is executed against the local underlying data source from which, the export schema is derived,

• The query result will be reorganized to fit the export schema, and returned to the user that has requested it. The Local Adaptation Layer reformats the results based on the local resources specifications (LRS) of the underlying data source.

Figure 6.14 illustrates the performing mechanism of the federated query processor and its interaction with the local query processor at each local database system. The federated query processor supports:

• Decomposition of the query arrived from an application (in common format) to a set of sub-queries, each to be sent to the corresponding underlying data source.

• Assembly of the partial returned sub-results into a common coherent format that can be understood by the requesting users/applications.

---

**Federated Query Decomposer**

The Federated Query Decomposer is a component of FQP, which decomposes a federated query to a set of sub-queries, each to be sent to the corresponding underlying data sources to be answered. The sub-queries are expressed using the common query language that is adopted at the federation layer. When a sub-query arrives at a site, the LAL at that site performs the translation of the sub-query to conform to the local query language and then it executes it.

**Results Assembler**

The Results Assembler is another component of FQP that combines the various sub-query results for every federated query, in the format of the common data model (CDM). Similar to the query translation process, the LAL at each site performs the transformation of the sub-query result from the local format (of its underlying data source) to the common format adopted at the federation layer. The Results Assembler then, merges the data produced by several local sites into a single format corresponding to the CDM.
6.2.3 Application of Database Standards and Middleware Solutions in GFI₂S

Considering the wide variety and heterogeneity of networked applications, any efficient solution for their integration/interoperation must involve the use of standard mechanisms for applications modeling, data structuring and information retrieval. Standard mechanisms ease the design and building of integration mechanisms for collaborative environments, and solve many barriers faced in the integration process. For instance, the use of emerging standards for application modeling (e.g. UML), for data structuring (e.g. ODL), for information retrieval (e.g. SQL or OQL), and for data exchange (e.g. OIF and XML) reduces the need for construction of individual data translation wrappers among integrated nodes, and facilitates the development of necessary federated query processor. However, similar to many others, in the database area, standards lag behind in supporting new features and extensions provided by certain commercial and research prototypes of database management systems. Most DBMS's extensions nowadays address the development of advanced constructs to better support the ever-growing requirements of emerging applications. The new constructs provided by these database systems address multimedia data types, object-orientation concepts, interoperation/integration facilities, distributed computing, etc.

The GFI₂S federated architecture, as conceived and described in this chapter, presents an open and flexible solution towards a generic approach for information exchange and data integration, applying standards to the extent possible, and suggesting their extension when standards are not available (see figure 6.15).

- On one hand, from the database development perspective, ODL is used in GFI₂S to support the portability of database schemas across conforming ODBMSs. OIF and XML are used to exchange objects between databases and provide database documentation. Simultaneously, independent access facilities among the data sources are supported through middleware solutions (e.g. ODBC, JDBC, and JAVA).

- On the other hand, from users, applications, and database accesses perspectives, the GFI₂S system targets a comprehensive solution, based on standard interfaces and languages (e.g. SQL, OQL, Java, and C++), which:
  - Provide most of the requested information for users and applications,
  - Facilitate the access to the heterogeneous and autonomous data sources,
  - Support flexible access to those underlying data sources based on standard technologies and via secure and reliable export schemas mechanisms,
  - Support multimedia information and large data sets.

The use of database standards and middleware solutions facilitate the exchange of information among different applications. Namely two aspects are emphasized. First, full and rich representation of the schema concepts, query language, and data representation are supported through the object-oriented database standards e.g. ODL, OQL, and OIF. Second, the user facilities and data transparency are supported through Web standards and Middleware solutions e.g. ODBC, Java, and XML. More details on the applicability of these standards and middleware to GFI₂S are included in Appendix A. Some of the problems that face the standardization process are also addressed and discussed within these sections.
6.2.4 GFI$_2$S in Action

The GFI$_2$S system is designed (and partially implemented) to better solve the information integration issues that were initially tackled in Chapters 3, 4, and 5. The flexible GFI$_2$S architecture can support a wide variety of collaborative environments, from a fully federated network of expert systems supporting distributed control applications (e.g. WaterNet), to a distributed Virtual Laboratory environment (e.g. VL), and even for brokerage of information from remote sites. This section describes one example operational case of the GFI$_2$S system for brokerage of information for distributed, heterogeneous, and autonomous sites. This example shows the quality of GFI$_2$S and illustrates the main benefits gained when deploying the GFI$_2$S approach. The example also demonstrates the integration of GFI$_2$S with universal access to data, data export, and other tools as explained in Chapter 5.

The example presented in figure 6.16 illustrates a flexible framework, which is based on the GFI$_2$S architecture. The three-tier (client/server) architecture adopted here for web services satisfies many information management requirements for the entire interoperation and collaboration tasks among distributed networked heterogeneous data sources and applications. It mainly (1) deploys the GFI$_2$S system, following the node-to-node federation approach, at the lower-tier; (2) a set of interfaces for data access and direct interaction among heterogeneous data sources using middleware and standard solutions, and (3) supports a number of defined users/applications accessing the integrated information system via these interfaces.

In more details, the global architecture presented in figure 6.16 is composed of the following components:

1. At the GFI$_2$S Federated Layer (lower-tier), the approach for integrating autonomous and heterogeneous data sources is addressed from two sides. On one hand, it develops a Local Adaptation Layer (LAL) for each data source, participating in the federated network. On the other hand, it builds the Node Federation Layer (NFL) on top of the conceptual data model (CDM) of the federated schema. The LAL components assure
proper communication between the local data sources and their federated layer. While, the NFL component supports the inter-connection of different information sources at the federated layer.

Additionally. The $\text{GFI}_2\text{S}$ architecture employs middleware and standards to provide standardized interfaces for all types of data (structured and unstructured), maximizing interoperability and reusability, while leaving the data either at the place where it is generated or where it is heavily used.

At the middle tier layer, three types of interfaces for local/remote data access are supported. The interfacing components facilitate the access to information for each type of authorized users in a secure and confident manner. The access to data is a dynamic process, which consists of direct interaction with heterogeneous data sources. The developed interfaces allow the access to the underlying heterogeneous data sources via the use of middleware and standard solutions (e.g. ODBC, JDBC, Java, and XML). The three main interfaces illustrated in 6.16 present the global access facilities to the underling heterogeneous database systems through $\text{GFI}_2\text{S}$, which mainly include:

- **Universal data access** and direct interaction with heterogeneous data sources using middleware solutions and standards.
- **Specific access facilities** (tailored interfaces) to data. These interface also dynamically interact with the remote data sources, but they are application dependent.
- **Full access** for administrators to heterogeneous remote data sources through the $\text{GFI}_2\text{S}$ tier.

These interfaces can also be used as a base for several enhancements. As such, from the Universal Data Access interface two variants were derived: the Safe/Reliable Data Export and the Flexible View-Based Access interface.
From the client tier, the four types of users/applications, which can access the data within such a system are:

* Database Administrator (DBA) that has full access to all data sources, and can update both the data and meta-data. The database administrator has the ability to access the data through standard tools or via specific access facilities.

* Users inside the organization (internal users/developers) are supported by a flexible data access interface, which is based on view definition for information visibility rights and security for access. Internal users may also have the ability to update the part of data of their responsibility.

* External users (collaborators) are supported by safe/reliable data export facilities through which, they can access information of their interest in an open and secure manner. In addition to the user friendly and flexible screen-based presentation of the requested information, this interface also supports the provision of data in several physical standard formats (OIF, XML, HTML, etc.) that can be uploaded at the user site. The safe and reliable data export interface only supports restricted read accesses.

* Specific applications that are clearly defined and are not expected to change are supported through specific application-dependent interfaces. These applications need self interaction with the database. information is requested from the database when needed and will be delivered at the requesting time.

### 6.3 Conclusion

The architecture of the Generic and Flexible Information Integration System (GFI₂S) is designed as a unified federated object-oriented layer that can support the integration of information among heterogeneous and autonomous data sources. Thus, supporting the information exchange and collaboration among heterogeneous and autonomous nodes. In this system, a data source can represent any database built using a commercial DBMS product, or any information source (e.g. a file system). The integrated schema, defined within the node federation layer of GFI₂S, provides users with a consolidated view of the information shared for the purpose of collaboration. The integrated schema is defined at the federated layer of every node in the network, while the data corresponding to it is distributed among the individual local/remote data sources. Thus, data in all of the local and remote nodes is accessible by others as if it belongs to a single local database.

The GFI₂S integration architecture is constituted of two main components: the Local Adaptation Layer (LAL) and the Node Federation Layer (NFL). This decomposition properly serves the need for sites’ interoperation and information sources integration. The GFI₂S architecture presents a flexible approach based on standards and middleware solutions, through which:

1. **The Local adaptation Layer (LAL)** at each node, defines the set of mechanisms and concepts that facilitate the access to underlying local data sources in their native format, and the control of their access rights.

2. **The Node Federation Level (NFL)** at each node, acts as a mediated common interface that sits between the local system and other external data sources participating in the federation.
The **GFI2S** federated architecture, as conceived and described in this chapter, presents an open and flexible solution towards a generic approach for information exchange and data integration:

- From the *database development perspectives*, ODL is used in the **GFI2S** to support the portability of database schemas across conforming ODBMSs, and OIF and XML are used to exchange objects between databases and provide database documentation. Simultaneously, independent access facilities among the data sources are supported through standards and middleware solutions (e.g., ODBC, JDBC, and JAVA).

- From the *user access perspectives*, the **GFI2S** system targets a comprehensive solution, based on standard interfaces and languages (e.g., SQL, OQL, Java, and C++), which facilitate the access to distributed, heterogeneous, and autonomous data sources. The flexible access to those underlying data sources is achieved through standard technologies and via secure and reliable export schemas mechanisms.

As such, the **GFI2S** federated approach allows the integration of databases and applications that are distributed, autonomous, and heterogeneous.