**Journal of Software Engineering and Applications, 2015, 8, 72-92**
Published Online February 2015 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2015.82009

Scientific Research Publishing

# Formalization of Federated Schema Architectural Style Variability

**Wilhelm Hasselbring**

Software Engineering Group, Department of Computer Science, Kiel University, Kiel, Germany
Email: hasselbring@email.uni-kiel.de

Received 25 January 2015; accepted 13 February 2015; published 15 February 2015

Copyright © 2015 by author and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY).
http://creativecommons.org/licenses/by/4.0/

Open Access

## Abstract

**Data integration requires managing heterogeneous schema information. A federated database system integrates heterogeneous, autonomous database systems on the schema level, whereby both local applications and global applications accessing multiple component database systems are supported. Such a federated database system is a complex system of systems which requires a well-designed organization at the system and software architecture level. A specific challenge that federated database systems face is the organization of schemas into a schema architecture. This paper provides a detailed, formal investigation of variability in the family of schema architectures, which are central components in the architecture of federated database systems. It is shown how the variability of specific architectures can be compared to the reference architecture and to each other. To achieve this, we combine the semi-formal object-oriented modeling language UML with the formal object-oriented specification language Object-Z. Appropriate use of inheritance in the formal specification, as enabled by Object-Z, greatly supports specifying and analyzing the variability among the studied schema architectures. The investigation also serves to illustrate the employed specification techniques for analyzing and comparing software architecture specifications.**

## Keywords

**Federated Database Systems, Software Architecture, Formal Specification, Software Product Families, Software Variability**

## 1. Introduction

A software product family consists of software systems that have some common functionality and some variable functionality [1]. The interest in software product families and software product lines emerged from the field of software reuse when developers and managers realized that they could obtain much greater reuse benefits by reusing software architectures instead of only reusing individual software components. The basic philosophy of

**How to cite this paper:** Hasselbring, W. (2015) Formalization of Federated Schema Architectural Style Variability. *Journal of Software Engineering and Applications*, **8**, 72-92. http://dx.doi.org/10.4236/jsea.2015.82009

software product families is reuse through the explicitly planned exploitation of commonalities of related products [2] and proper management of *variability* in software systems [3] [4].

Product development in software product families is organized into two stages: domain engineering and application engineering [5]. The idea behind this approach to product engineering is that the investments require to develop the reusable artifacts during domain engineering are outweighed by the benefits of deriving the individual products during application engineering.

For large, complex software systems, the design of the overall system structure (the software architecture) is a central problem. The *architecture* of a software system defines that system in terms of components and connections among those components [6] [7]. It is not the *design* of that system which is more detailed. The architecture shows the correspondence between the requirements and the constructed system, thereby providing some rationale for the design decisions. This level of design has been addressed in a number of ways including informal diagrams and descriptive terms, module interconnection languages, and frameworks for systems that serve the needs of specific application domains. An architecture embodies decisions about quality properties. It represents the earliest opportunity for evaluating those decisions. Furthermore, reusability of components and services depends on how strongly coupled they are with other components in the system architecture. Performance, for instance, depends largely upon the complexity of the required coordination, in particular when the components are distributed via some network.

Similar to civil engineering where the engineer knows that a house includes a roof at the top and a cellar at the bottom etc., with compiler construction the software engineer should know that a compiler contains a pipeline including lexical analysis, parsing, semantic analysis, and code generation. As yet, this is not the case for all areas of software development. In the present paper, we investigate the schema architecture of federated database systems to make some progress in this domain enabling formal analysis and comparison of specific federated schema architectures. The schema architecture is a central part in the architecture of federated database systems. We study the family of federated schema architectural styles in this paper.

Constructing a formal specification requires some effort that should be justified. The primary motivation for formalizing federated schema architecture styles is to enable analysis and comparison of specific schema architectures for evaluating and selecting appropriate architectural variations. Usually, formal specifications consist of interleaved passages of formal, mathematical text and informal prose explanation. We propose a three-level interleaving of formality in the specification:

- Informal prose explanation (illustrated with examples);
- Semi-formal object-oriented modeling (we use the UML for this purpose);
- Rigorous formal specification (we use Object-Z for this purpose).

An important goal is to obtain a well structured formal specification. Formal specifications are often criticized by practitioners, because it is hard to comprehend them. To some extent, the problems are due to missing (visual) structure in the specification. With our approach, the object-oriented diagrams provide an overview of the formal specification, and a first level of (semi) formalization. In our view, graphical specifications are appropriate for providing an overview, while textual specifications are appropriate for providing details.

Our contribution is a detailed, formal investigation of variability in the family of some representative schema architectures. These schema architectures are central components in the architecture of their federated database systems. It is shown how the variability of specific architectures can be compared to the reference architecture and to each other. The investigation also serves to illustrate the employed specification techniques for analyzing and comparing software architecture specifications.

In Section 2, we discuss software product families and their architectures, as far as relevant for the present paper. Section 3 gives an overview of federated database systems in general and the reference schema architecture in particular, before the reference schema architecture is formalized in Section 4. This formalization allows for a precise comparison with specific architectures in Section 5. Sections 6 and 7 discuss related work and draw some conclusions.

## 2. Software Product Families and Their Architectures

The study of software architecture has evolved from the seminal work of Perry & Wolf [8], Garlan & Shaw [6], and others. Architecture Description Languages (ADLs) have emerged to lend formal rigor to architecture representation [9]. Despite more than two decades of research on architecture description languages [9], in

industrial practice software architectures are usually described informally or semi-formally with diagrams using boxes, circles and lines together with accompanying prose. The prose explains the diagrams and provides some rationale for the chosen architecture. Typical examples are the above-mentioned *pipeline* architecture for the various phases of a compiler or a *client-server* architecture for distributed information systems. Such figures often give an intuitive picture of the system's construction, but the semantics of the components and their connections/interactions may be interpreted by different people in different ways (due to the informality). Some specific advantages of formality in software architecture description may be summarized as follows:

- Software architectures become amenable to analysis and evaluation [10]. This helps to evaluate architectures and to guide in the selection of architectural variations as solutions to specific problems.
- Software architectures can be a basis for *design reuse* [11] [12], provided that the individual elements of the architectural descriptions are defined independently and in a precise way. Re-usable architectures give designers a *blueprint* in development by helping them avoid typical design errors.
- Software architectures support improved program understanding as a basis for system evolution if its specification is well understood: retaining the designer's intention about a system organization should help maintainers preserve the system's design integrity [13] [14].
- Formality can allow prototyping for early design evaluation [15].
- Testing may be supported by deriving test plans from formal architectural descriptions [16] [17].
- Proper tool support for designing and analyzing software architectures becomes possible [18].

Reference architectures play an important role in domain engineering. *Domain engineering* is an activity for building reusable components, whereby the systematic creation of domain models and architectures is addressed. Domain engineering aims at supporting *application engineering* which uses the domain models and architectures to build concrete systems. The emphasis is on reuse and product lines. The Domain-Specific Software Architecture (DSSA) engineering process was introduced to promote a clear distinction between domain and application requirements [19]. A Domain-Specific Software Architecture consists of a domain model and a reference architecture. The DSSA process consists of domain analysis, architecture modeling, design and implementation stages as illustrated in **Figure 1** [5]. The DSSA process concentrates on gathering architectural information about specific application domains. Reference architectures are the basic structures used to build systems in a product line. The domain model characterizes the *problem space*, while the *reference architecture* addresses the *solution space* in domain engineering.

Software architectures address the attributes indicated above for single systems. DSSA capture architectural commonality of multiple, related systems, *i.e.*, systems within the same domain. DSSA are central to domain-specific reuse, in that they provide a framework for creating assets and constructing systems within a domain. Domain engineering also allows for product-line development, which seeks to achieve reuse across a family of systems [20] [21]. Federated database management systems can be regarded as such a family of systems.

## 3. Federated Database Systems

A *database system* consists of a database management system and one or more databases that it manages. In many application areas, data is distributed over a multitude of heterogeneous, autonomous database systems.

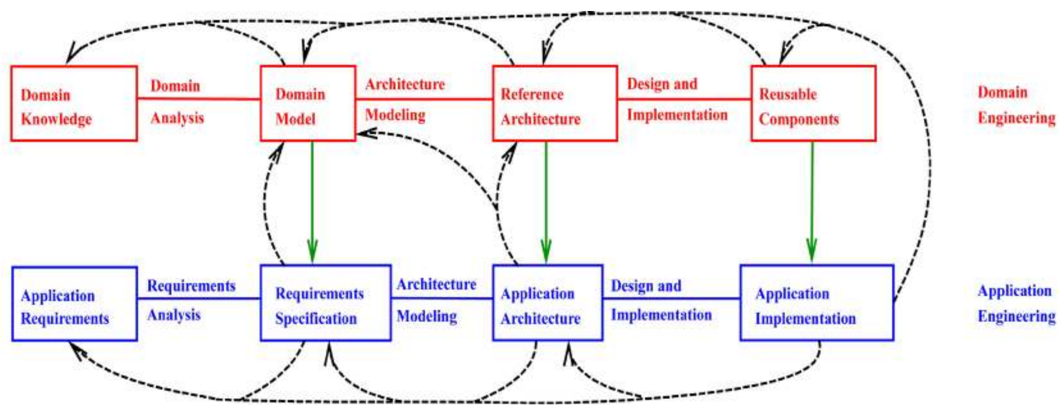



**Figure 1.** The DSSA engineering process [5].

These systems are often isolated and an exchange of data among them is not easy. On the other hand, support for dynamic exchange of data is required to improve the business processes. Global access to several local systems and consistently replicating/moving data among systems are typical requirements.

A federated database system is an integration of such autonomous database systems, where both local applications and global applications accessing multiple database systems are supported [22]. A federated database system is a complex *system of systems* which requires a well designed organization at the software architecture level. The work on federated database systems centers around three issues: autonomy, heterogeneity, and distribution.

- Federated database systems are characterized by a controlled and sometimes limited integration of **autonomous** database systems. Often, there are conflicts between requirements of integration and autonomy.
- **Heterogeneity** is caused by different database management and operating systems, as well as the design autonomy among component database systems.
- In the case of federated database systems, much of the **distribution** is due to the existence of individual database systems *before* a federated database system is built (legacy systems).

Federated database systems may be distinguished as being *tightly* and *loosely* coupled [22]. In the case of loosely coupled federated database systems, each component site builds its own federated schema by integrating its local schema with the export schemas of some other component sites. Loose coupling promotes integration and interoperability via *multidatabase query languages* which allow uniform access to all component database systems [23]. It does not require the existence of integrated schemas, leaving many responsibilities, such as dealing with multiple representations of data and resolving semantic mismatch, to the programmer of component database systems. Tight coupling requires schema integration. The present paper discusses tightly federated database systems with a schema-based integration architecture.

## 3.1. The System Architecture of Federated Database Systems

Let us start with a look at the typical overall system architecture of federated database systems, which is displayed in **Figure 2**. In a federated database system, both global applications and local applications are supported. The local applications remain autonomous to a great extent, but must restrict their autonomy to some extent to participate in the federation. Global applications can access multiple local database systems through the federation layer. The federation layer can also control global integrity constraints such as data value dependencies across multiple component database systems. The dependencies among the schemas in the component database systems are specified on the federation layer within a *schema architecture*, which is a kind of *metadata* describing the local systems, their correspondences, and what is offered to global applications.

To achieve a division of labor between system components, database *wrappers* should be connected to the component database systems to simplify the kernel. The database wrappers transform the data between the local data models and the canonical data model of the federation layer (see below). **Figure 2** illustrates this division of labor between federation layer kernel and wrappers. The local database management systems of the component databases see the wrappers as local applications. This approach allows for a "separation of concerns" between the federation kernel and the component wrappers. The responsibility for monitoring and announcing changes in component database systems is delegated from the kernel of the federation layer to the wrappers for the individual component database systems. This way, the kernel of the federation layer may see the component database systems as active database systems [24]. An active database system is an extended conventional database system which has the capability to monitor predefined situations (situations of interest) and to react with defined actions. "Separation of concerns" is an important principle to manage complexity in software engineering [25].

## 3.2. The Five-Level Reference Schema Architecture for Federated Database Systems

As mentioned in the previous subsection, the schema architecture is a central component of the federation layer. A problem that all federated database system face is the organization of schemas in such a schema architecture. A reference architecture for schemas is useful to clarify the various issues and choices within those complex federated systems. It provides the framework in which to understand, categorize and compare different architectural options for developing specific systems. A reference architecture can be used as a basis for analysis and comparison of such-like architectures, as will be shown later in Section 5.

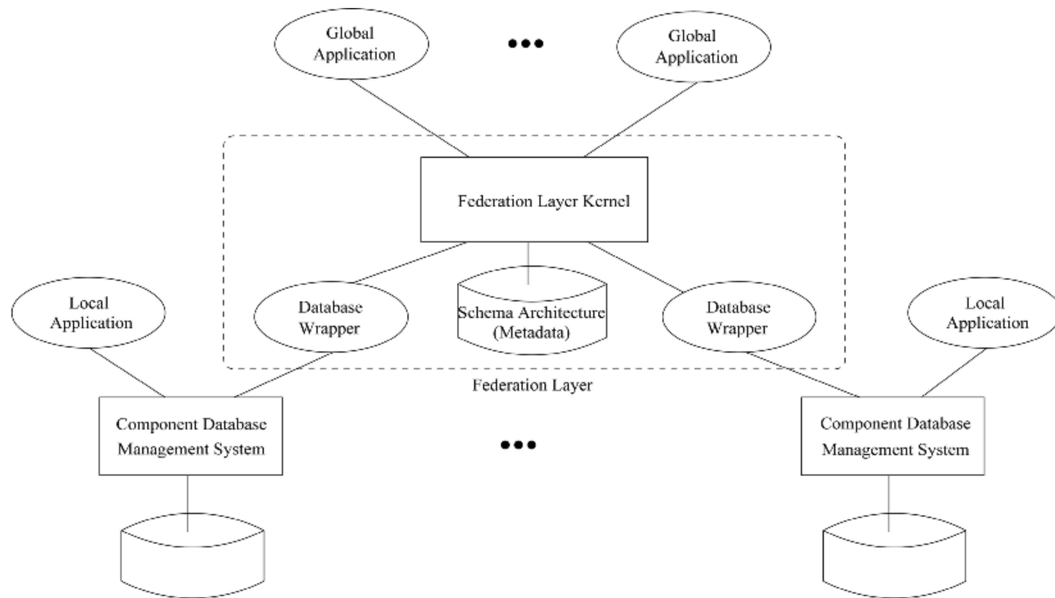



**Figure 2.** The general system architecture of federated database systems with database wrappers as mediators between component database systems and federation layer kernel.

For federated database systems, the traditional three-level schema architecture [26] must be extended to support the dimensions of distribution, heterogeneity, and autonomy. The generally accepted reference architecture for schemas in federated database systems is presented in [22] and, in the same form, in [27] where approaches to object-orientation in multidatabase systems are surveyed. The diagram in **Figure 3** illustrates this schema architecture which presents, apart from the dots that indicate repetition, a possible configuration of schemas in a federated database system. The different schema types are:

**Local Schema**: A Local Schema is the conceptual schema of a component database system which is expressed in the (native) data model of that component database system.

**Component Schema**: A Component Schema is a Local Schema transformed into the (canonical) data model of the federation layer. Object-oriented data models such as ODMG are often employed as canonical data models [28].

**Export Schema**: An Export Schema is derived from a Component Schema and defines an interface to the local data that is made available to the federation.

**Federated Schema**: When Exported Schemas are semantically heterogeneous, it is necessary to integrate them using another level. A Federated Schema on this higher level is the result of the integration of multiple Export Schemas; thus, providing an integrated view. Top-down or bottom-up integration strategies may be applied [29] [30].

**External Schema**: An External Schema is a specific view on a Federated Schema or on a Local Schema. External Schemas may base on a specific data model different to the canonical data model. Basically, External Schemas serve as specific interfaces for applications. External Schemas serve global applications if filtered from a Federated Schema, while serving local applications when filtered from a Local Schema.

This schema architecture, which is managed by the federation layer, specifies the dependencies/correspondences among the individual schemas. The database wrappers in **Figure 2** need to know the corresponding Local and Component Schemas, and the mapping between the native and canonical data models. The upper levels in the canonical data model are managed by the federation kernel, which may offer External Schemas in some native data models to global applications. Two important features of the schema architecture are how autonomy is preserved and how access control is managed [22]. Autonomy is preserved by dividing the administrative control among some component database system administrators and a federation system administrator. The Local, Component and Export Schemas are controlled by the component system administrators. The federation system administrator defines and manages the Federated Schemas and the External Schemas which are related to the Federated Schemas. Note, however, that one person can take on the role of several system administrators.

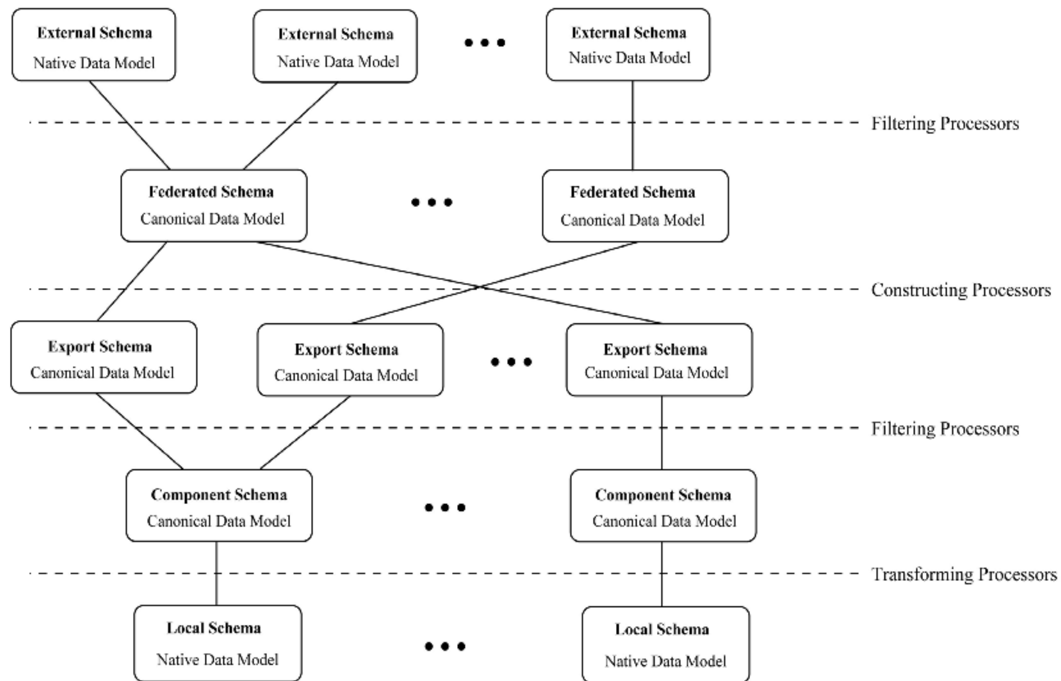



**Figure 3.** The five-level schema architecture as presented by Sheth & Larson [22]. We indicate the corresponding processor types between the levels my means of dotted lines.

The accompanying text in [22] explains that several options and constraints in the schema architecture are available, some of which are:

- Any number of External Schemas can be defined.
- Any number of Federated Schemas can be defined.

A federated database system with multiple federations allows the tailoring of the use of the federated database system with respect to multiple classes of global federation users with different data access requirements.

- Schemas on all levels, except the Local and Federated Schemas, are optional.

Note that a schema architecture which consists of just one Federated Schema and some Local Schemas still concurs with the five-level schema architecture of [22]. The other levels contain no schemas in this case.

- A component database system can participate in more than one federation and continue the operation of local applications. Thus, Local Schemas may be mapped to several Component, Export, and also External Schemas.

These constraints are not defined formally, particularly not explicit in **Figure 3** which is meant to illustrate the reference architecture. It is necessary to read the informal descriptions in [22] very carefully to comprehend all these options and constraints. In Section 4, a formal specification will be presented which defines these options and constraints with mathematical rigor.

As reported in [31], this reference schema architecture of [22] is generally accepted as the basic structure in federated database systems or at least for comparison with other specific architectures. However, several modifications have been proposed, as shall be discussed in Section 5. However, this reference provides a basic architecture for managing the complexity of resolving data heterogeneity among component systems.

## 4. Formalization of the Reference Schema Architecture

Software architectures hide many of the implementation details which are not relevant to understanding the important differences among alternate architectures. A formal specification language such as Object-Z [32] [33] is well-suited for concentrating on the essential concerns and neglecting irrelevant details through abstraction. The graphical, semi-formal UML specification will be employed for providing an overview of the specified software architectures, as well as the structure for the formal Object-Z specification itself. As mentioned in the introduction, the specification of the federated schema architecture style is presented in three steps:

Step 0: Informal prose explanation: done in Section 3.2. (with examples);
Step 1: Semi-formal modeling: see Section 4.1. (with the UML);
Step 2: Rigorous formal specification: see Section 4.2. (with Object-Z).

The goal is to obtain a well structured formal specification. With our approach, the object-oriented UML models provide an overview of the formal Object-Z specification and a first level of (semi) formalization.

### 4.1. First Step: The Semi-Formal UML Specification

The Unified Modeling Language (UML) is a graphical modeling language that has been standardized by the Object Management Group [34]. **Figure 4** displays our first step towards formally specifying the reference schema architecture for federated database systems using the UML notation for class diagrams. In this model, some of the constraints and options for the architecture, which were discussed in Section 3, are defined by means of the *multiplicities* at the associations among classes. The example schema architecture in **Figure 3** can be regarded as an instance of the class model in **Figure 4**, which defines a *metamodel* for schemas and their associations in the reference schema architecture. In **Figure 3**, *multiplicity* of schemas was indicated by means of some dots.

A more detailed account of the model in **Figure 4** is given as follows. Rectangles are the UML symbols for classes. Inheritance for specialization and generalization is shown in UML as a solid-line path from the subclass to the superclass, with a hollow triangle at the end of the path where it meets the superclass. In the UML, multiplicities for associations are specified through numerical ranges at the association links. The default multiplicity is 1. If the multiplicity specification comprises a single asterisk, then it denotes the unlimited non-negative integer range (zero or many). The arrows attached to the association names indicate the direction for reading the names which are annotations to associations. For instance, the association between Local Schema and Component Schema in **Figure 4** specifies that each Component Schema is transformed from exactly one Local Schema, but each Local Schema can be transformed into multiple Component Schemas, when the corresponding component database participates in more than one federation.

Some additional constraints, which are not specified in the UML model, are the following.

- Federated Schemas are required to be integrated from *at least one* Local, Component or Export Schema, *i.e.*, Federated Schemas must be connected to the ground and not levitate.
- Each Export Schema is filtered from at least one Component or Local Schema.
- Each External Schema is derived from either one Federated or one Local Schema. External Schemas which are directly derived from Local Schemas are used for local applications.

Those additional constraints cannot be specified *graphically* within this class diagram. It is necessary to specify them textually by means of additional informal prose and/or the UML Object Constraint Language [34] [35]. The Object Constraint Language allows some predicate logic to be specified. However, the class diagram in **Figure 4** is a *semi-formal* specification as the *semantics* of the UML notation has not been specified formally.

To summarize, this UML diagram has already specified many details of the schema architecture that were not explicitly specified in the illustration of [22] in **Figure 3**. However, still several details are missing in the graphical model. Consider, for instance, the constraint that Federated Schemas are required to be integrated from *at least one* Local, Component or Export Schema. Furthermore, the semantics of the UML notation is not fully formally specified. Therefore, we take the second step to further formalize the schema architecture by means of a formal specification language.

### 4.2. Second Step: The Formal Object-Z Specification

Object-Z [32] [33] is an extension of the formal specification language Z [36] to facilitate specification in an object-oriented style. It is a conservative extension in the sense that the existing syntax and semantics of Z are retained in Object-Z. Object-Z specifications are strongly typed, what means that all defined elements have to be used within a correct typing context.

The following Object-Z specifications are presented in a bottom-up way, what means that classes are defined before they are used by other classes. Similar to Z, Object-Z obeys the principle of *definition before use*. Such bottom-up development is typical for object-oriented techniques [37]. Similarly, federated database systems themselves can be developed bottom-up starting with the integration of existing component database systems, but also top-down starting with the development of a global federation layer and the subsequent integration of

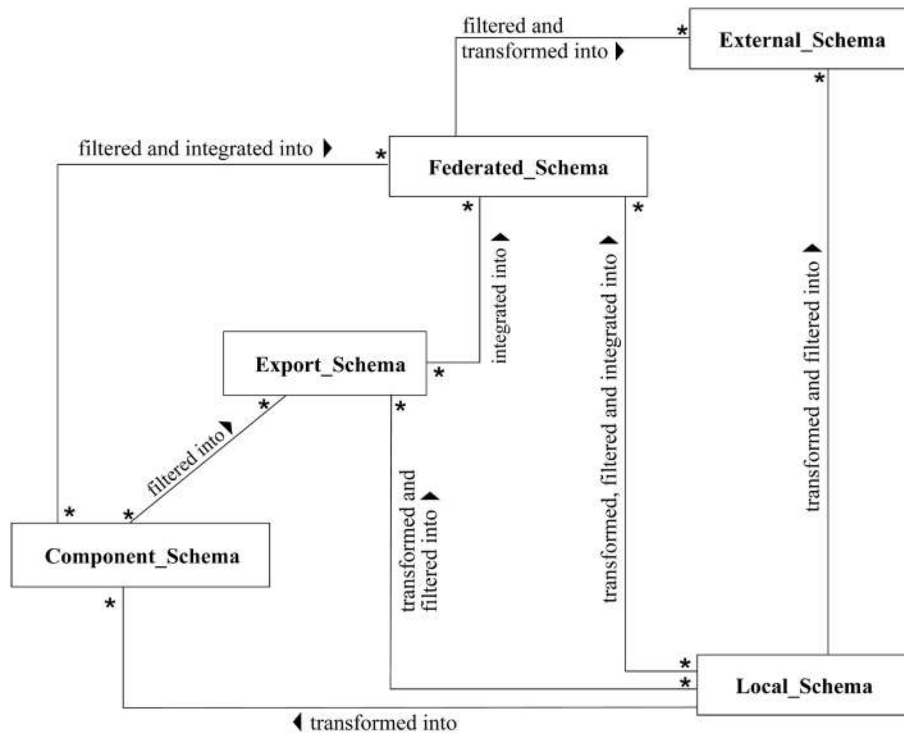



**Figure 4.** Modeling the five-level reference schema architecture for tightly coupled federated database systems as a UML class diagram.

(possibly new) component database systems [29], and even with a combination of both strategies [30].

Specifying the schema architecture of federated database systems, data models for describing schemas are basic.

[DATA_MODEL]

Basic type definitions introduce new types in Z and Object-Z, whose internal structure is not relevant for the specification. In our specification, we abstract from the details of data models since these details are not relevant on the schema *architecture* level.

The different data models used in the schema architecture are then defined in an axiomatic definition.

$$| \ Native_Data_Model, Canonical_Data_Model : DATA_MODEL$$

It is important to find the right level of abstraction for describing software architectures. On the architectural level of the schema architecture, there is nothing more to be said about the data models. This would be a concern of detailed design. Let the Object-Z class.

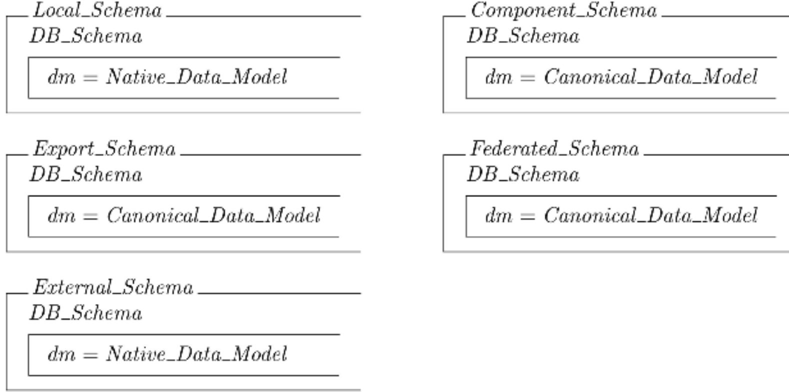
*DB_Schema*

$dm : DATA_MODEL$

denote possible database schemas without specifying the concrete data models. Because we are—at the architecture level—not interested in the detailed content of a database schema, the class $DB_Schema$ has no additional attributes or operations.

The class $DB_Schema$ is an *abstract* or *deferred* class [37] which will be specialized through inheritance into *concrete* classes for the different schema types later on. Object-Z provides no explicit syntactical constructs for distinguishing abstract classes from concrete classes. We assume that object instances of the class

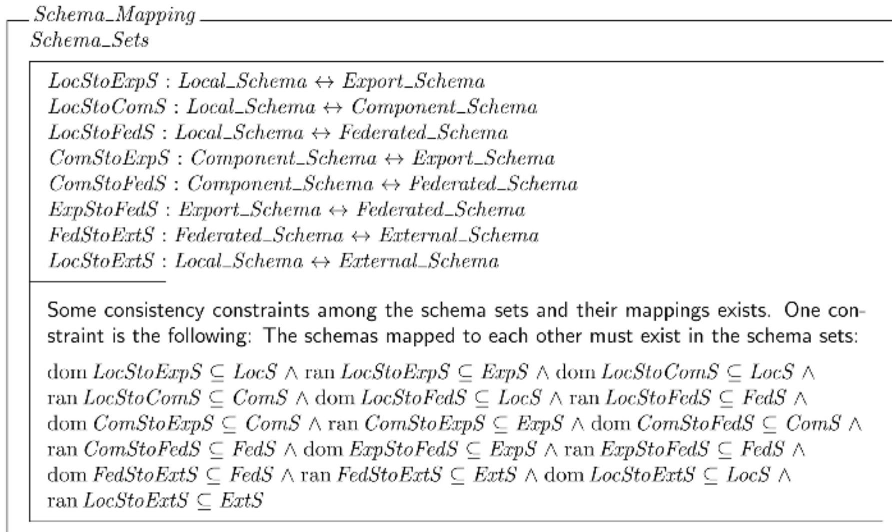
*DB_Schema* contain all relevant information on schemas in databases in the federation. This assumption is made on the grounds of simplification: we do not want to have to overspecify the Object-Z class at this stage of the architecture specification. The five different schema types are then defined as subclasses of *DB_Schema* as follows.



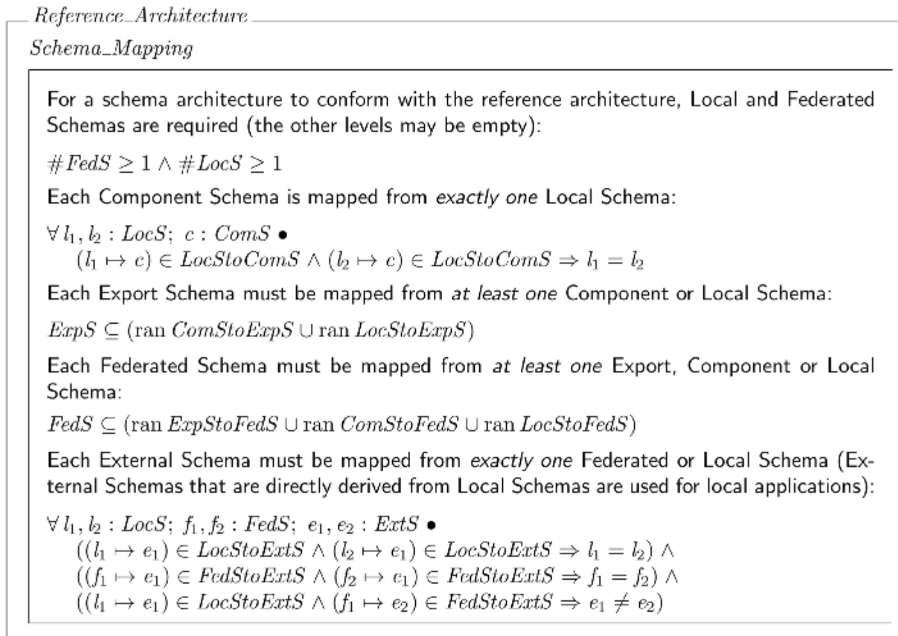

If we had used plain Z [36], *basic type definitions* would have been introduced instead of the above class specification as it has been done above with the basic type *DATA_MODEL*. However, with this Object-Z specification of the abstract class *DB_Schema* we are able to supply additional details later on in the specification through inheritance. Such an undertaking would not be possible with only using plain Z's basic type definitions.

As illustrated in **Figure 3** and **Figure 4**, a schema architecture consists of schemas on the five levels as defined in the class *Schema_Sets*.

The class *Schema_Mapping* represents the mappings among those schemas in *Schema_Sets*. It *inherits* the schema sets and adds the mappings (incremental specification by means of object-oriented inheritance).



So far, we defined the components of a schema architecture which are sets of schemas and mappings among the schemas in these sets. Now, the question arises: How to define the schema architecture itself? Particularly, the question is *how* to specify the reference architecture such that it can be compared to the architectures of specific systems. We define the reference architecture as a specialization of *Schema_Sets* that satisfies several constraints.



*Reference_Architecture*

*Schema_Mapping*

For a schema architecture to conform with the reference architecture, Local and Federated Schemas are required (the other levels may be empty):

$\# FedS \geq 1 \wedge \# LocS \geq 1$

Each Component Schema is mapped from *exactly one* Local Schema:

$\forall l_1, l_2 : LocS;\ c : ComS \bullet$
$(l_1 \mapsto c) \in LocStoComS \wedge (l_2 \mapsto c) \in LocStoComS \Rightarrow l_1 = l_2$

Each Export Schema must be mapped from *at least one* Component or Local Schema:

$ExpS \subseteq (\operatorname{ran} ComStoExpS \cup \operatorname{ran} LocStoExpS)$

Each Federated Schema must be mapped from *at least one* Export, Component or Local Schema:

$FedS \subseteq (\operatorname{ran} ExpStoFedS \cup \operatorname{ran} ComStoFedS \cup \operatorname{ran} LocStoFedS)$

Each External Schema must be mapped from *exactly one* Federated or Local Schema (External Schemas that are directly derived from Local Schemas are used for local applications):

$\forall l_1, l_2 : LocS;\ f_1, f_2 : FedS;\ e_1, e_2 : ExtS \bullet$
$((l_1 \mapsto e_1) \in LocStoExtS \wedge (l_2 \mapsto e_1) \in LocStoExtS \Rightarrow l_1 = l_2) \wedge$
$((f_1 \mapsto e_1) \in FedStoExtS \wedge (f_2 \mapsto e_1) \in FedStoExtS \Rightarrow f_1 = f_2) \wedge$
$((l_1 \mapsto e_1) \in LocStoExtS \wedge (f_1 \mapsto e_2) \in FedStoExtS \Rightarrow e_1 \neq e_2)$

As mentioned before, most of these constraints cannot be specified graphically in the UML model of Section 4.1. After formalizing the reference schema architecture of [22], we formalize and compare some specific federated schema architectures to the reference and to each other in the following section.

## 5. Formalization and Comparison of Some Specific Systems

In this section, we relate the concepts of the reference architecture to those realized in some specific federated database management systems. The purpose is not to survey these systems comprehensively, but to show how the reference schema architecture can be compared to the schema architectures of various federated database systems. Such a representation aims to support the task of studying and comparing these systems.

We will exemplarily formalize and compare the IRO-DB, IBM InfoSphere, FOKIS, and BLOOM schema architectures in the following subsections to wrap up with a comparison in Section 5.5.

### 5.1. The IRO-DB Schema Architecture

The IRO-DB (Interoperable Relational and Object Databases) [38] project developed a set of tools to achieve interoperability of pre-existing relational databases and new object-oriented databases. One of the main goals of the project was the provision of a path for integrating the relational database technology to object-oriented database technology.

In the IRO-DB architecture, it is emphasized that there may be multiple Federated Schemas and not one global integrated schema. Therefore, Federated Schemas are called *interoperable schemas* in their schema architecture. Component Schemas are not used, instead Local Schemas are always directly transformed into Export Schemas. Also, External Schemas are not used; instead individual interoperable schemas can directly be specified for specific requirements of global applications. **Figure 5** displays the semi-formal UML model for the IRO-DB schema architecture. We intentionally leave the layout in the style of the UML model for the reference schema architecture for easier visual comparison with **Figure 4**.

Essentially, the IRO-DB schema architecture is a subset of the reference architecture. We specify the specific

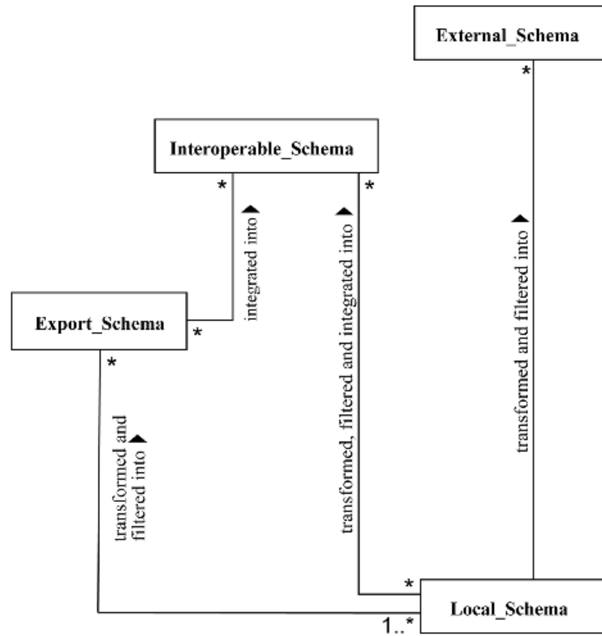



**Figure 5.** Modeling the IRO-DB schema architecture as a UML class diagram.

architectural constraints of the IRO-DB schema architecture as a specialization of the Object-Z class *Schema_Mapping* in two steps. First, we define a basic schema architecture as follows.

*Basic_Architecture*

*Schema_Mapping*

As in the reference architecture, in the basic schema architecture both Local and Federated Schemas are required:

$\#FedS \geq 1 \wedge \#LocS \geq 1$

As opposed to the reference architecture, in the basic schema architecture there exist no Component Schemas and no mappings from/to them:

$0 = \#ComS = \#LocStoComS = \#ComStoExpS = \#ComStoFedS$

Each Export Schema must be mapped from *at least one* Local Schema:

$ExpS \subseteq \operatorname{ran} LocStoExpS$

Each Federated Schema must be mapped from *at least one* Export or Local Schema:

$FedS \subseteq (\operatorname{ran} ExpStoFedS \cup \operatorname{ran} LocStoFedS)$

In the second step, the IRO-DB schema architecture is specified as a specialization of the basic architecture with the following constraints.

*IRO_DB_Architecture*

FedS is renamed to InteropS while inheriting the Schema_Mapping:

$Basic_Architecture[InteropS / FedS]$

As opposed to the reference architecture, in the IRO-DB schema architecture no mappings from Federated to External Schemas are supported:

$0 = \#FedStoExtS$

This two-step specialization allows for reuse of the basic architecture for similar systems, as we will see in the

following subsection.

### 5.2. The IBM InfoSphere Schema Architecture

IBM InfoSphere Federation Server [39] is a commercial federated database system that provides global access to multiple heterogeneous databases, formerly known as IBM DB2 Universal DataJoiner [40]. It provides transparent access to tables at remote databases through user defined aliases (so-called *nicknames*) that can be accessed as if they were local tables. The InfoSphere Federation Server is also a fully functional relational database system, based on IBM DB2. Among other features, the DB2 query optimizer is available at the federation layer and inserts, updates and deletes are redirected to the integrated, local databases.

Schema integration with InfoSphere Federation Server is accomplished via nicknames. As an example, nicknames to external Oracle database and Sybase database tables are defined with InfoSphere as follows.

```
CREATE NICKNAME O_EMP FOR ORACLE.USER_X.EMP

CREATE NICKNAME S_OFFICE FOR SYBASE.USER_Y.OFFICE
```

These nicknames correspond to Export Schemas in the reference architecture. Views over these nicknames can then be defined via standard SQL.

```
CREATE VIEW FEDERATED_SCHEMA
    AS SELECT O_EMP.EMPNAME, S_OFFICE.OFFICENO
       FROM O_EMP, S_OFFICE
       WHERE O_EMP.EMPNO = S_OFFICE.EMPNO
```

This federated view corresponds to the Federated Schemas in the reference architecture. **Figure 6** displays the UML model for the InfoSphere schema architecture.

The specific architectural constraints of the InfoSphere schema architecture are again specified as a specialization of the Object-Z class *Basic_Architecture* as follows.

*InfoSphere_Architecture*

Export Schemas are Nicknames and Federated Schemas are views:

$Basic_Architecture[Nicknames / ExpS, FedView / FedS]$

Thus, apart from some renaming, the InfoSphere schema architecture is just a basic architecture.

### 5.3. The FOKIS Schema Architecture

This section discusses a federated schema architecture which has been designed according to the specific requirements of integrating replicated information among heterogeneous components of hospital information systems [41]. It is rather obvious that the reference schema architecture of [22] has been designed primarily to support global access to the component database systems, only secondarily to support data replication. However, the reference schema architecture is a good framework for resolving the syntactic and semantic conflicts among heterogeneous schemas and for integrating the schemas. Therefore, the reference schema architecture has been extended in FOKIS by Publish, Subscribe and Publish/Subscribe distinction for Export Schemas to adequately support the algorithms for updating replicated information with the FOKIS system. This distinction does not exist in the reference architecture.

**Figure 7** displays the class diagram for this extended schema architecture using the UML notation. The distinct specializations of Export Schemas replace the Export Schemas in the reference architecture.

| *Publish_Schema* | *Subscribe_Schema* | *PubSub_Schema* |
|---|---|---|
| *Export_Schema* | *Export_Schema* | *Export_Schema* |

Export Schema becomes an abstract class [37] in this architecture: in actual instances of the schema architec-

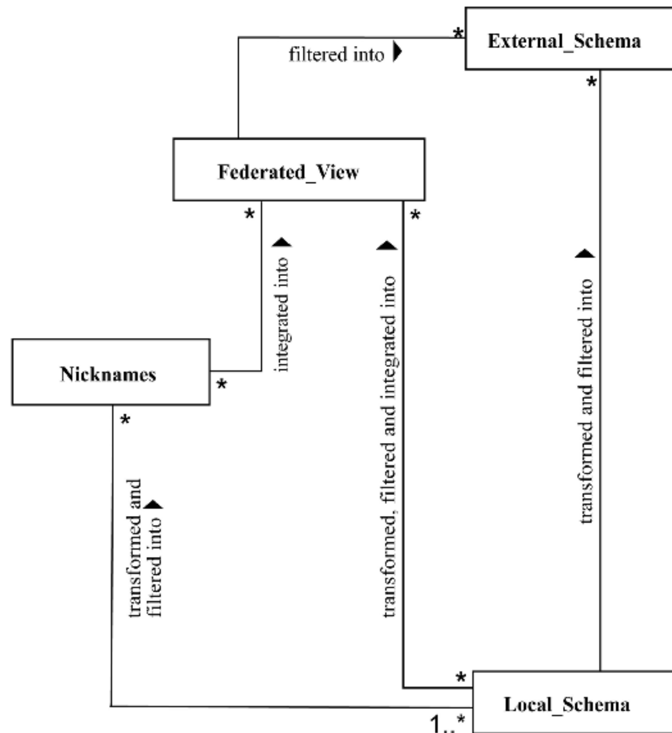



Figure 6. Modeling the InfoSphere schema architecture as a UML class diagram.

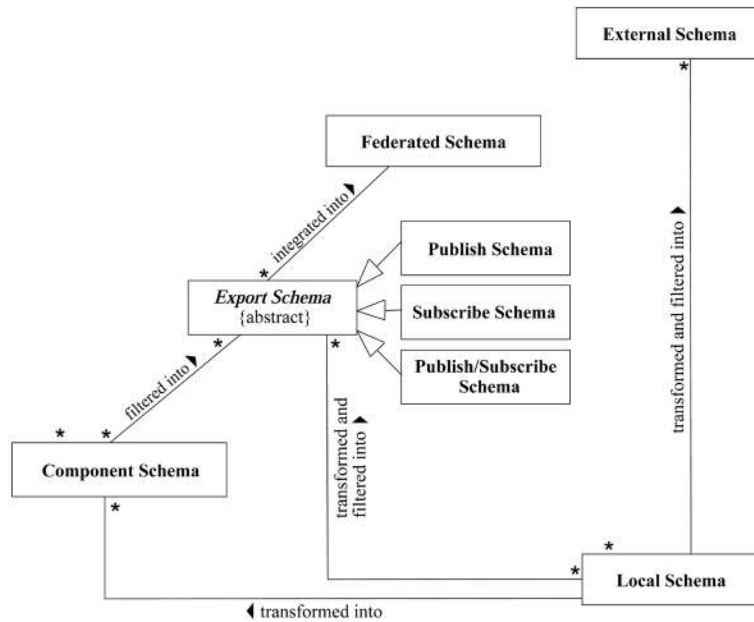
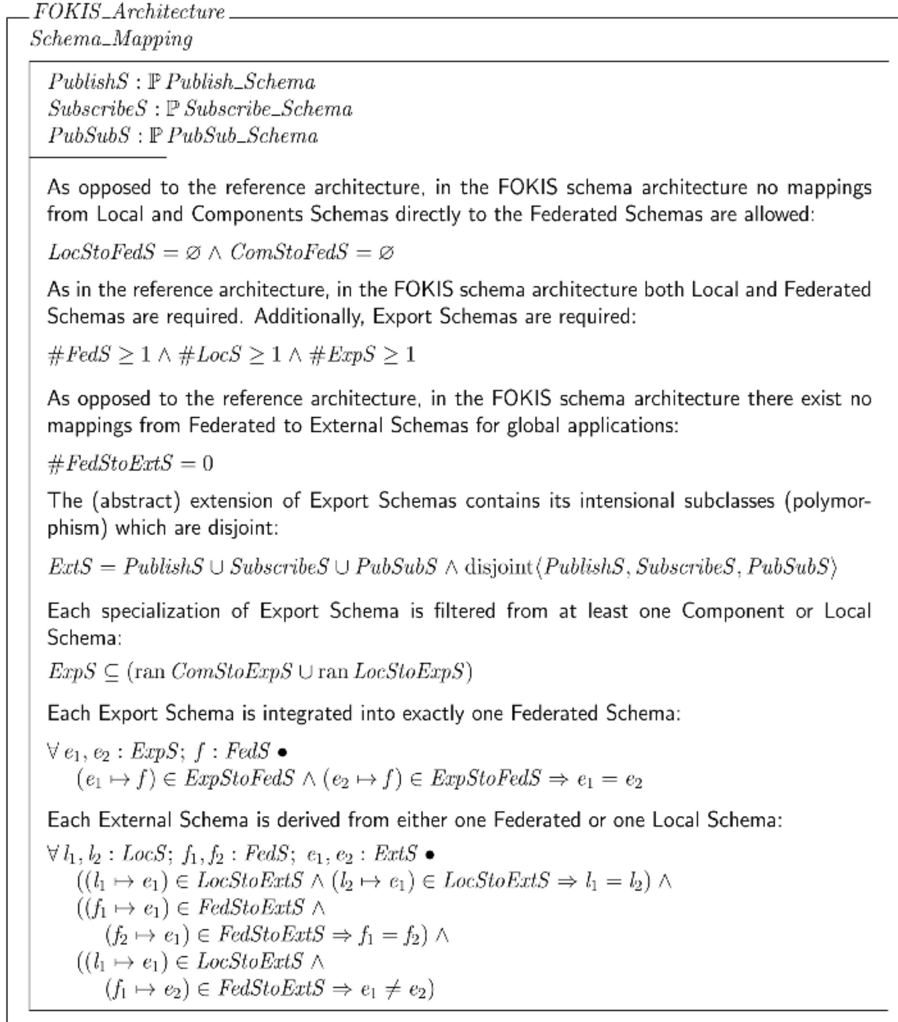



Figure 7. Modeling the FOKIS schema architecture as a UML class diagram.

ture only the subclasses of Export Schema are instantiated. The concrete classes Publish Schema, Subscribe Schema, and Publish/Subscribe Schema inherit all associations from *Export Schema*. There will be no instances (schemas) of the abstract class *Export Schema* in an instantiated schema architecture.

Specifying a Subscribe Schema in this architecture is a subscription to change notifications for the corresponding data. Publish Schemas specify data to be exported to other systems. Publish/Subscribe Schemas define

data to be both imported and exported. The schema types determine the change algorithms for integration of replicated information in an event-based interaction architecture as discussed in [41]. The FOKIS schema architecture is then specified in Object-Z as follows.


*FOKIS_Architecture*

*Schema_Mapping*

$PublishS : \mathbb{P}\ Publish_Schema$
$SubscribeS : \mathbb{P}\ Subscribe_Schema$
$PubSubS : \mathbb{P}\ PubSub_Schema$

As opposed to the reference architecture, in the FOKIS schema architecture no mappings from Local and Components Schemas directly to the Federated Schemas are allowed:

$LocStoFedS = \varnothing \wedge ComStoFedS = \varnothing$

As in the reference architecture, in the FOKIS schema architecture both Local and Federated Schemas are required. Additionally, Export Schemas are required:

$\#FedS \geq 1 \wedge \#LocS \geq 1 \wedge \#ExpS \geq 1$

As opposed to the reference architecture, in the FOKIS schema architecture there exist no mappings from Federated to External Schemas for global applications:

$\#FedStoExtS = 0$

The (abstract) extension of Export Schemas contains its intensional subclasses (polymorphism) which are disjoint:

$ExtS = PublishS \cup SubscribeS \cup PubSubS \wedge \text{disjoint}\langle PublishS, SubscribeS, PubSubS\rangle$

Each specialization of Export Schema is filtered from at least one Component or Local Schema:

$ExpS \subseteq (\text{ran}\ ComStoExpS \cup \text{ran}\ LocStoExpS)$

Each Export Schema is integrated into exactly one Federated Schema:

$\forall e_1, e_2 : ExpS;\ f : FedS \bullet$
$\quad (e_1 \mapsto f) \in ExpStoFedS \wedge (e_2 \mapsto f) \in ExpStoFedS \Rightarrow e_1 = e_2$

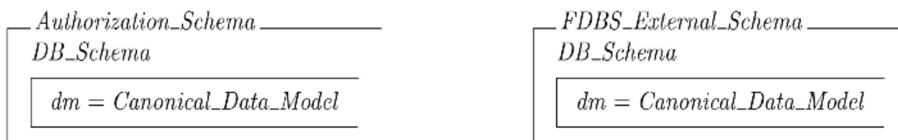
Each External Schema is derived from either one Federated or one Local Schema:

$\forall l_1, l_2 : LocS;\ f_1, f_2 : FedS;\ e_1, e_2 : ExtS \bullet$
$\quad ((l_1 \mapsto e_1) \in LocStoExtS \wedge (l_2 \mapsto e_1) \in LocStoExtS \Rightarrow l_1 = l_2) \wedge$
$\quad ((f_1 \mapsto e_1) \in FedStoExtS \wedge$
$\qquad (f_2 \mapsto e_1) \in FedStoExtS \Rightarrow f_1 = f_2) \wedge$
$\quad ((l_1 \mapsto e_1) \in LocStoExtS \wedge$
$\qquad (f_1 \mapsto e_2) \in FedStoExtS \Rightarrow e_1 \neq e_2)$

## 5.4. The BLOOM Schema Architecture

The BLOOM approach to federated database systems consist of a schema architecture with several federated schemas and a functional architecture [42] [43]. The functional architecture in BLOOM includes the components needed to build the federation and the main modules of the execution architecture, see also Section 3.1. The reference schema architecture has been extended in BLOOM to deal with security aspects not well addressed previously. A new seven-level schema architecture framework has been developed in BLOOM. **Figure 8** displays the UML model for this BLOOM schema architecture.

In particular, the authorization schema level with a filter mechanism to federated external schemas has been added to the reference architecture to address security concerns on the schema level.


*Authorization_Schema*

*DB_Schema*

$dm = Canonical_Data_Model$

*FDBS_External_Schema*

*DB_Schema*

$dm = Canonical_Data_Model$

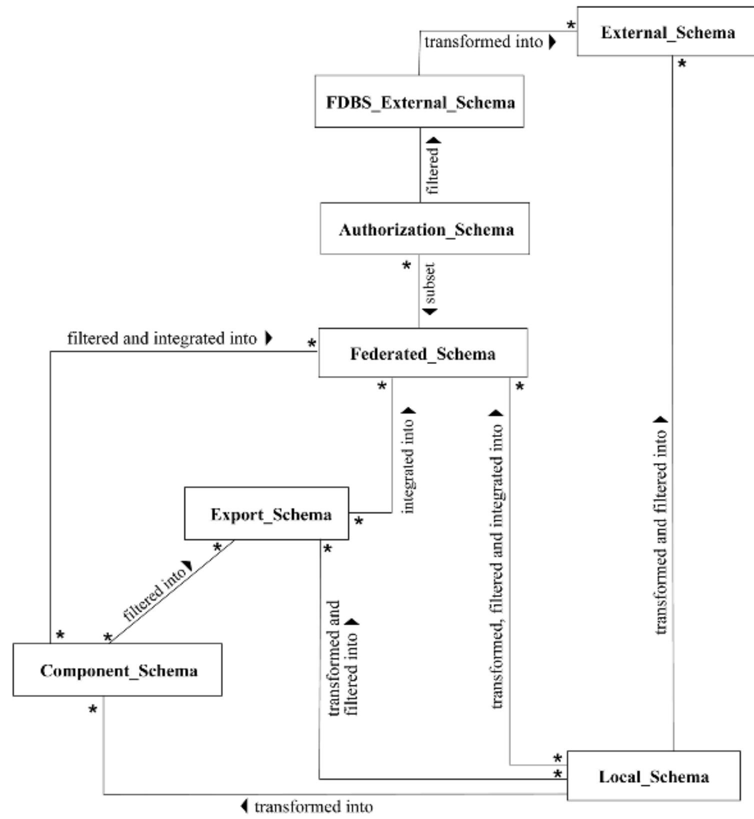



**Figure 8.** Modeling the BLOOM seven-level schema architecture as a UML class diagram.

With these additional schema levels, we can specify the BLOOM schema architecture.

$BLOOM_Architecture$

$Reference_Architecture$

$AuthorizationS : \mathbb{P}\, Authorization_Schema$
$FDBS_ExtS : \mathbb{P}\, FDBS_External_Schema$

In Object-Z, $\rightarrow$ specifies total functions and $\rightarrowtail\!\!\!\!\twoheadrightarrow$ bijective functions:

$FedStoAuthS : Federated_Schema \rightarrow Authorization_Schema$
$AuthStoFDBSExtS : Authorization_Schema \rightarrowtail\!\!\!\!\twoheadrightarrow FDBS_External_Schema$
$FDBSExtStoExtS : FDBS_External_Schema \rightarrow External_Schema$

The schemas mapped to each other must exist in the schema sets:

$\mathrm{dom}\, FedStoAuthS \subseteq FedS \wedge \mathrm{ran}\, FedStoAuthS \subseteq AuthS \wedge$
$\mathrm{dom}\, AuthStoFDBSExtS \subseteq AuthS \wedge \mathrm{ran}\, AuthStoFDBSExtS \subseteq FDBS_ExtS \wedge$
$\mathrm{dom}\, FDBSExtStoExtS \subseteq FDBS_ExtS \wedge \mathrm{ran}\, FDBSExtStoExtS \subseteq ExtS$

The additional schema levels must not be empty:

$\# FDBS_ExtS \geq 1 \wedge \# AuthS \geq 1$

Each Authorization Schema is mapped from exactly one Federated Schema:

$\forall a_1, a_2 : AuthS;\ f : FedS \bullet$
$\quad ((a_1 \mapsto f) \in FedStoAuthS \wedge (a_2 \mapsto f) \in FedStoAuthS) \Rightarrow a_1 = a_2$

## 5.5. Comparison of the Schema Architectures

**Figure 9** illustrates the structure of our Object-Z specifications as a UML class diagram. This model is not part of the specifications for the schema architectures, it documents the structure of these specifications.

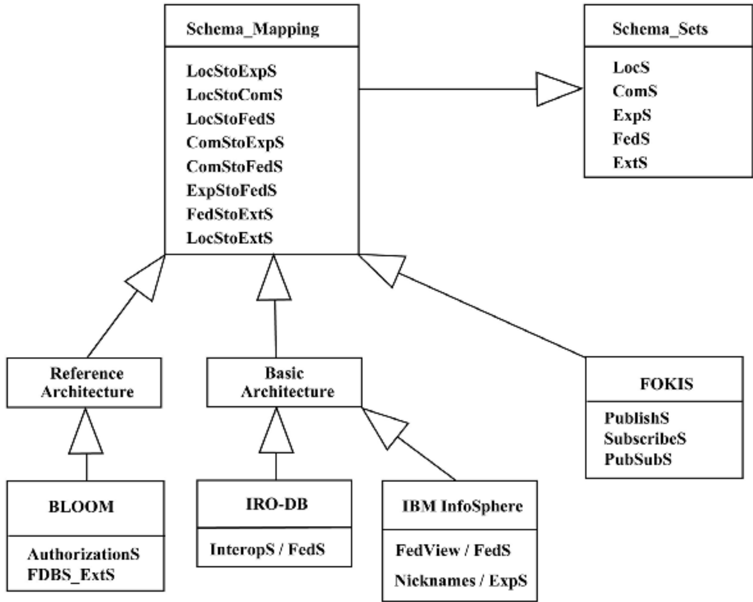



**Figure 9.** The structure of our Object-Z specification modeled as a UML class diagram.

The super classes *Schema_Sets* and *Schema_Mapping* define the general components of a schema architecture as sets of schemas and mappings among the schemas in these sets. Both the reference architecture and the architectures of the selected systems are direct or indirect specializations of the class *Schema_Mapping*. The different ways of specializing the *Schema_Mapping* serve as one means to compare the specific systems with the reference architecture and with each other on a high level.

The IRO-DB and IBM InfoSphere schema architectures are minimal subsets of the *Schema_Mapping*, they share the *Schema_Mapping*. Some schema types have different names in these architectures. The IRO-DB and IBM's InfoSphere schema architectures can be seen as some kind of minimal approach, which imposes several restrictions, if we compare it to the reference architecture. The FOKIS schema architecture has been designed according to the specific requirements of integrating replicated information among heterogeneous information systems. The mechanisms for publishing and subscribing to specific data items required some extensions to the generic part, while retaining its fundamental structure. The BLOOM schema architecture extends the reference architecture with several structures for security mechanisms.

The different ways of specializing and constraining the super classes serve as a means to compare the specific systems with the reference architecture and with each other. The specialization hierarchy already offers a gross overview of the similarities and differences. The detailed comparison can be done by taking a close look at the formal Object-Z specification. However, there exists an important difference between architectural comparisons on the UML and the Object-Z levels: On the UML level we compare individual class diagrams to each other. On the Object-Z level the comparison is based on the amount of customization of the schema mappings required for constructing the specific schema architectures. This allows for a more detailed comparison.

## 6. Related Work

### 6.1. Formalization of Architectural Styles and Design Patterns

Formalization of architectural styles aims to allow formal checks of conformance between architecture and implementation to predict the impact of changes, to formally reason about a system's architectural description, or to make rigorous comparisons among different architectural descriptions [44]. Various approaches to formalizing architectural styles have been proposed [45]-[47]. Within the overall framework of domain engineering, our work presented in this paper addresses the reference and application schema architecture construction. Specific schema architectures for individual federated database systems have been constructed with the reference schema architecture in mind. The formal comparison allows for a detailed analysis of similarities and

differences among the architectural variations. Reuse on the design level plays an important role in this context [12]. On the programming level, reuse is usually accomplished by means of high-level programming language constructs, function libraries, or object-oriented class frameworks. On the design level, design patterns and established software architectures are essential. Design patterns [48] are "micro-architectures" while software architectures are more coarse-grained designs. A design pattern describes a family of solutions to a recurring problem. Patterns form larger wholes like pattern languages or handbooks when woven together so as to provide guidance for solving complex problem sets. Patterns express the understanding gained from practice in software design and construction. The patterns community catalogs useful design fragments and the context that guides their use. Approaches to formalizing patterns address pattern detection on source code [49] or supporting the work with patterns in integrated development environments [50] [51]. Our work intends to formalize variability analysis on the architectural level.

### 6.2. Formalizing the Unified Modeling Language

Because the UML comprises several different notations with no formal semantics attached to the individual diagrams, it is not possible to apply rigorous automated analysis on them. Several approaches introduce mappings between metamodels describing UML and some formal languages [52]. For instance, France *et al.* [53] employ Z, and Snook & Butler [54] employ the B method for formalizing the UML. These approaches enable the construction of rules for transforming UML models into specifications in some formal language. The resulting specifications derived from UML diagrams enable, for instance, execution through simulation or analysis via model checking. The goal of these approaches is to assign formal semantics to UML models. Our approach is to employ the UML for alleviating the comprehension of formal specifications (in Object-Z in our case). Similar to our work, Kim & Carrington [55] propose an integrated framework with UML and Object-Z for developing a precise and understandable specification. This work is motivated by the assessment that formal specification techniques provide a systematic approach to produce a precise and analyzable software specification. However, the notations provided by most formal specification techniques are often difficult to use and understand. Kim & Carrington intend to overcome these limitations by combining graphical specification techniques with formal specification techniques to show that an integrated approach is beneficial for both graphical and formal specification techniques. Representing formal specifications using appropriate diagrams can improve understanding of the formal specifications. So, the work of Kim & Carrington is similar to our approach. We are applying it to product-line variability analysis.

### 6.3. Software Variability Analysis

Software product families were defined by Parnas [56] as a set of programs to constitute a family whenever it is worthwhile to study programs from the set by first studying the common properties of the set and then determining the special properties of the individual family members. The SCV (Scope, Commonality and Variability) analysis method, for instance, is an approach that provides a systematic way of identifying and organizing a product family to be created [57]. SCV analysis is one approach to defining a family by identifying commonalities, *i.e.* assumptions that are true for all family members, variabilities, *i.e.* assumptions about what can vary among family members, and common terminology for the family. Variability analysis with SCV contains guidelines and requirements to assist in the identification and analysis of variabilities, which may include a cost-benefit analysis. Ramachandran & Allen [58] employ semi-formal data-flow context diagrams and use-case diagrams for variability analysis, while we are combining class diagrams with a formal specification language. Metzger *et al.* [3] formalize the documentation of variability in software product lines. They emphasize the difference between software variability and product-line variability. We formalize product-line variability for the family of federated schema architectures, while Metzger *et al.* formalize feature diagrams that are used for documenting variability. Another formalization of feature models may be found in [59].

## 7. Conclusions and Future Work

The highest costs in software development is generally in system maintenance and the addition of new features. If it is done early on, architectural evaluation can reduce that cost by revealing a design's implications [10]. This, in turn, can lead to an early detection of errors and to the most predictable and cost-effective modifications to

the system over its life cycle. There are several reasons to consider an architectural view.

1) Architecture is often the first design artifact that represents decisions on how requirements of all types are to be achieved. As the manifestation of early design decisions, it represents design decisions that are hardest to change and hence most deserving of careful consideration.

2) An architecture is also the key artifact in successfully engineering a product line. Product-line engineering is the disciplined, structured development of a family of similar systems with less effort, expense, and risk than it would be incurred if each system were developed independently.

3) The architecture is usually the first artifact to be examined when a programmer (particularly a maintenance programmer) who is unfamiliar with the system begins to work on it.

The individual federated database management systems studies in this paper were developed independently, as far as we know. However, the published papers on these systems all refer to the reference architecture of [22]; thus, all systems were inspired by that basic reference. Our comparison investigates the similarities and differences of the realized specific architectures. The formal specifications clarified several aspects of the specific schema architectures that were not explicit in the informal descriptions. Appropriate use of inheritance in the formal specification, as enabled by Object-Z, greatly supports specifying and analyzing the variability among the studied schema architectures. The different ways of specializing the general parts via inheritance in the specification serve as a means to compare the specific systems with the reference architecture and with each other.

However, some training in formal notations is required to understand the formal specification. To address this, our formal specification is accompanied with informal prose explanation and a semi-formal specification employing the established object-oriented modeling notation UML. The object-oriented specification with the combination of the UML and Object-Z allowed for an incremental, well-structured construction of the formal specifications.

A formal model such as the presented one provides a precise, mathematically based description of a family of systems. The model attempts to capture a class of systems that is otherwise understood only idiomatically, and to expose the essential characteristics of that family while hiding unnecessary details. One of the benefits of doing this is that we can analyze various properties of systems designed in this style. Such an analysis could be supported by an animation of the formal specification as presented in [60].

Formal specifications have the advantage of precision and unambiguousness, what is not the case for those object-oriented modeling techniques which are usually used in industrial settings. On the other hand, the textual mathematical notation of formal specifications is only intelligible with a solid mathematical background, while object-oriented models are usually understandable for the average programmer. The combination of a semi-formal object-oriented modeling language with a formal object-oriented specification language aims at exploiting the advantages of both approaches. An important contribution of this paper is the formalization of the schema architecture for federated database systems which can be used as a guide by software engineers who intend to build a specific federated database system. Future work could aim at building a product line for developing similar federated systems.

Federated database management systems are already a research topic for many years, and database systems in general are a well-understood research area with a reasonable formal underpinning. These established fundamentals helped with formalizing and comparing the architectures. In the future, we intend to apply the presented approach to less established areas, such as variability in service-centric integration architectures [61] and in peer-to-peer architectures [62].

## References


[1] Gomaa, H. (2004) Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley, Boston.

[2] Deelstra, S., Sinnema, M. and Bosch, J. (2004) Experiences in Software Product Families: Problems and Issues during Product Derivation. In: Nord, R.L., Ed., *Software Product Lines*, *Lecture Notes in Computer Science Volume* 3154, Springer-Verlag, Berlin, 165-182.

[3] Metzger, A., Pohl, K., Heymans, P., Schobbens, P. and Saval, G. (2007) Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis. *Proceedings of the* 15*th IEEE International Requirements Engineering Conference*, Delhi, 15-19 October 2007, 243-253.

[4] Galster, M., Weyns, D., Tofan, D., Michalik, B. and Avgeriou, P. (2014) Variability in Software Systems—A Syste-

matic Literature Review. *IEEE Transactions on Software Engineering*, **40**, 282-306. http://dx.doi.org/10.1109/TSE.2013.56

[5] Hasselbring, W. (2002) Component-Based Software Engineering. In: Chang, S.K., Ed., *Handbook of Software Engineering and Knowledge Engineering*, World Scientific Publishing, Singapore, 289-305. http://dx.doi.org/10.1142/9789812389701_0013

[6] Shaw, M. and Garlan, D. (1996) Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, Englewood Cliff.

[7] Taylor, R.N., Medvidovic, N. and Dashofy, E.M. (2009) Software Architecture: Foundations, Theory, and Practice. John Wiley and Sons, Hoboken.

[8] Perry, D. and Wolf, A. (1992) Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, **17**, 40-52. http://dx.doi.org/10.1145/141874.141884

[9] Medvidovic, N. and Taylor, R.N. (2000) A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, **26**, 70-93. http://dx.doi.org/10.1109/32.825767

[10] Clements, P., Kazman, R. and Klein, M. (2001) Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley, Boston.

[11] Frakes, W.B. and Kang, K. (2005) Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering*, **31**, 529-536. http://dx.doi.org/10.1109/TSE.2005.85

[12] Shaw, M. (1995) Architectural Issues in Software Reuse: It's Not Just the Functionality, It's the Packaging. *Software Engineering Notes*, **20**, 3-6.

[13] Bennett, K.H. and Rajlich, V.T. (2000) Software Maintenance and Evolution: A Roadmap. *Proceedings of the Conference on the Future of Software Engineering*, Limerick, 4-11 June 2000, 73-87.

[14] Müller, H., Wong, K. and Tilley, S.R. (1995) Dimensions of Software Architecture for Program Understanding. *Proceedings of the International Workshop on Software Architecture*, Dagstuhl, 20-24 February 1995, 1-4.

[15] Luckham, D.C., Kenney, J.J., Augustin, L.M., Vera, J., Bryan, D. and Mann, W. (1995) Specification and Analysis of System Architecture Using Rapide. *IEEE Transactions on Software Engineering*, **21**, 336-355. http://dx.doi.org/10.1109/32.385971

[16] Muccini, H., Bertolino, A. and Inverardi, P. (2004) Using Software Architecture for Code Testing. *IEEE Transactions on Software Engineering*, **30**, 160-171. http://dx.doi.org/10.1109/TSE.2004.1271170

[17] Bertolino, A., Corradini, F., Inverardi, P. and Muccini, H. (2000) Deriving Test Plans from Architectural Descriptions. *Proceedings of the* 22*th International Conference on Software Engineering*, Limerick, 4-11 June 2000, 220-229.

[18] Shaw, M., De Line, R., Klein, D.V., Ross, T.L., Young, D.M. and Zelesnik, G. (1995) Abstractions for Software Architecture and Tools to Support Them. *IEEE Transactions on Software Engineering*, **21**, 314-335. http://dx.doi.org/10.1109/32.385970

[19] Taylor, R.N., Tracz, W.J. and Coglianese, L. (1995) Software Development Using Domain-Specific Software Architectures. *ACM SIGSOFT Software Engineering Notes*, **20**, 27-38. http://dx.doi.org/10.1145/217030.217034

[20] Macala, R., Stuckey, L. and Gross, D. (1996) Managing Domain-Specific, Product-Line Development. *IEEE Software*, **13**, 57-67. http://dx.doi.org/10.1109/52.493021

[21] Dikel, D., Kane, D., Ornburn, S., Loftus, W. and Wilson, J. (1997) Applying Software Product-Line Architecture. *Communications of the ACM*, **30**, 49-55.

[22] Sheth, A. and Larson, J. (1990) Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, **22**, 183-236. http://dx.doi.org/10.1145/96602.96604

[23] Tresch, M. and Scholl, M.H. (1994) A Classification of Multi-Database Languages. *Proceedings of the* 3*rd International Conference on Parallel and Distributed Information Systems*, Austin, 28-30 September 1994, 195-202. http://dx.doi.org/10.1109/PDIS.1994.331716

[24] Widom, J. and Ceri, S., Eds. (1996) Active Database Systems—Triggers and Rules for Advanced Database Processing. Morgan Kaufmann, San Francisco.

[25] Ghezzi, C., Jazayeri, M. and Mandrioli, D. (2003) Fundamentals of Software Engineering. 2nd Edition, Prentice Hall, Englewood Cliffs.

[26] Date, C.J. (2004) An Introduction to Database Systems. 8th Edition, Addison-Wesley, Reading.

[27] Pitoura, E., Bukhres, O. and Elmagarmid, A. (1995) Object Orientation in Multidatabase Systems. *ACM Computing Surveys*, **27**, 141-195. http://dx.doi.org/10.1145/210376.210378

[28] Roantree, M., Murphy, J. and Hasselbring, W. (1999) The OASIS Multidatabase Prototype. *ACM SIGMOD Record*, **28**, 97-103. http://dx.doi.org/10.1145/309844.310066

[29] van den Heuvel, W.-J., Hasselbring, W. and Papazoglou, M. (2000) Top-Down Enterprise Application Integration with Reference Models. *Australian Journal of Information Systems*, **8**, 126-136.

[30] Hasselbring, W. (2002) Web Data Integration for E-Commerce Applications. *IEEE Multimedia*, **9**, 16-25. http://dx.doi.org/10.1109/93.978351

[31] Conrad, S., Eaglestone, B., Hasselbring, W., Roantree, M., Saltor, F., Schönhoff, M., Strässler, M. and Vermeer, M.W.W. (1997) Research Issues in Federated Database Systems: Report of EFDBS' 97 Workshop. *SIGMOD Record*, **26**, 54-56. http://dx.doi.org/10.1145/271074.271089

[32] Duke, R., Rose, G. and Smith, G. (1995) Object-Z: A Specification Language Advocated for the Description of Standards. *Computer Standards and Interfaces*, **17**, 511-533. http://dx.doi.org/10.1016/0920-5489(95)00024-O

[33] Smith, G. (1999) The Object-Z Specification Language. Kluwer Academic Publishers, Boston.

[34] OMG Unified Modeling Language Version 2.5, December 2013. http://www.omg.org/spec/UML/

[35] Warmer, J. and Kleppe, A. (2003) The Object Constraint Language: Getting Your Models Ready for MDA. 2nd Edition, Addison-Wesley Professional, Boston.

[36] Spivey, J.M. (1992) The Z Notation: A Reference Manual. 2nd Edition, Prentice-Hall, Englewood Cliff.

[37] Meyer, B. (1997) Object-Oriented Software Construction. 2nd Edition, Prentice-Hall, Englewood Cliff.

[38] Gardarin, G., Finance, B. and Fankhauser, P. (1997) Federating Object-Oriented and Relational Databases: The IRO-DB Experience. *Proceedings of the* 2nd *IFCIS International Conference on Cooperative Information Systems*, Kiawah Island, 24-27 Jun 1997, 2-13.

[39] IBM Corporation (2013) Data Virtualization: Delivering On-Demand Access to Information throughout the Enterprise. http://ibm.com/software/products/us/en/ibminfofedeserv/

[40] Venkataraman, S. and Zhang, T. (1998) Heterogeneous Database Query Optimization in DB2 Universal Data Joiner. *Proceedings of the* 24*th International Conference on Very Large Data Bases*, New York, 24-27 August 1998, 685-689.

[41] Hasselbring, W. (1997) Federated Integration of Replicated Information within Hospitals. *International Journal on Digital Libraries*, **1**, 192-208. http://dx.doi.org/10.1007/s007990050016

[42] Rodrguez, E., Oliva, M., Saltor, F. and Campderrich, B. (1997) On Schema and Functional Architectures for Multilevel Secure and Multiuser Model Federated DB Systems. *Proceedings of the International CAiSE*'97 *Workshop Engineering Federated Database Systems*, Barcelona, 16-17 June 1997, 93-104.

[43] Saltor, F., Campderrich, B., Rodriguez, E. and Rodriguez, L.C. (1996) On Schema Levels for Federated DB Systems. *Proceedings of the Parallel and Distributed Computing Systems*, Reno, 19-21 June 1996, 766-771.

[44] Abowd, G., Allen, R. and Garlan, D. (1995) Formalizing Style to Understand Descriptions of Software Architecture. *ACM Transactions on Software Engineering and Methodology*, **4**, 319-364.http://dx.doi.org/10.1145/226241.226244

[45] Bernardo, M. and Inverardi, P., Eds. (2003) Formal Methods for Software Architectures, Volume 2804 of Lecture Notes in Computer Science. Springer, Berlin.

[46] Pahl, C., Giesecke, S. and Hasselbring, W. (2007) An Ontology-Based Approach for Modelling Architectural Styles. *Proceedings of the* 1*st European Conference on Software Architecture*, Aranjuez, 24-26 September 2007, 60-75.

[47] Pahl, C., Giesecke, S. and Hasselbring, W. (2009) Ontology-Based Modelling of Architectural Styles. *Information and Software Technology*, **51**, 1739-1749. http://dx.doi.org/10.1016/j.infsof.2009.06.001

[48] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) Design Patterns—Elements of Reusable Object-Oriented Software. Addison Wesley, Reading.

[49] Albin-Amiot, H., Cointe, P., Guéhéneuc, Y.-G. and Jussien, N. (2001) Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together. *Proceedings of the* 16*th IEEE International Conference on Automated Software Engineering*, San Diego, 26-29 November 2001, 166-173.

[50] Mak, J.K.H., Choy, C.S.T. and Lun, D.P.K. (2004) Precise Modeling of Design Patterns in UML. *Proceedings of the International Conference on Software Engineering*, Edinburgh, 23-28 May 2004, 252-261.

[51] Guennec, A.L., Sunyé, G. and Jézéquel, J.-M. (2000) Precise Modeling of Design Patterns. In: Evans, A., Kent, S. and Selic, B., Eds., *UML* 2000—*The Unified Modeling Language*, *Volume* 1939 *of Lecture Notes in Computer Science*, Springer, Berlin, 482-496.

[52] Mc Umber, W.E. and Cheng, B.H.C. (2001) A General Framework for Formalizing UML with Formal Languages. *Proceedings of the* 23*rd International Conference on Software Engineering*, Toronto, 12-19 May 2001, 433-442.

[53] France, R., Evans, A., Lano, K. and Rumpe, B. (1998) The UML as a Formal Modeling Notation. *Computer Standards & Interfaces*, **19**, 325-334. http://dx.doi.org/10.1016/S0920-5489(98)00020-8

[54] Snook, C. and Butler, M. (2006) Uml-B: Formal Modeling and Design Aided by UML. *ACM Transactions on Software Engineering and Methodology*, **15**, 92-122. http://dx.doi.org/10.1145/1125808.1125811

[55] Kim, S.-K. and Carrington, D. (2000) An Integrated Framework with UML and Object-Z for Developing a Precise and Understandable Specification: The Light Control Case Study. *Proceedings of the 7th Asia-Pacific Software Engineering Conference*, Singapore, 5-8 December 2000, 240-248.

[56] Parnas, D. (1976) On the Design and Development of Program Families. *IEEE Transactions on Software Engineering*, **2**, 1-9. http://dx.doi.org/10.1109/TSE.1976.233797

[57] Coplien, J., Hoffman, D. and Weiss, D. (1998) Commonality and Variability in Software Engineering. *IEEE Software*, **15**, 37-45. http://dx.doi.org/10.1109/52.730836

[58] Ramachandran, M. and Allen, P. (2005) Commonality and Variability Analysis in Industrial Practice for Product Line Improvement. *Software Process*: *Improvement and Practice*, **10**, 31-40. http://dx.doi.org/10.1002/spip.212

[59] Czarnecki, K., Helsen, S. and Eisenecker, U. (2005) Formalizing Cardinality-Based Feature Models and Their Specialization. *Software Process*: *Improvement and Practice*, **10**, 7-29. http://dx.doi.org/10.1002/spip.213

[60] Hasselbring, W. (1994) Animation of Object-Z Specifications with a Set-Oriented Prototyping Language. *Proceedings of the 8th Z User Meeting*, Cambridge, 29-30 June 1994, 337-356.

[61] Pahl, C., Hasselbring, W. and Voss, M. (2009) Service-Centric Integration Architecture for Enterprise Software Systems. *Journal of Information Science and Engineering*, **25**, 1321-1336.

[62] Bischofs, L., Giesecke, S., Gottschalk, M., Hasselbring, W., Warns, T. and Willer, S. (2006) Comparative Evaluation of Dependability Characteristics for Peer-to-Peer Architectural Styles by Simulation. *Journal of Systems and Software*, **79**, 1419-1432. http://dx.doi.org/10.1016/j.jss.2006.02.063

Scientific Research Publishing www.scirp.org

Scientific Research Publishing (SCIRP) is one of the largest Open Access journal publishers. It is currently publishing more than 200 open access, online, peer-reviewed journals covering a wide range of academic disciplines. SCIRP serves the worldwide academic communities and contributes to the progress and application of science with its publication.

**Other selected journals from SCIRP are listed as below. Submit your manuscript to us via either submit@scirp.org or Online Submission Portal.**