Top-Down vs. Bottom-Up Engineering of Federated Information Systems

Wilhelm Hasselbring

INFOLAB, Tilburg University
PO Box 90153, 5000 LE Tilburg, The Netherlands
hasselbring@kub.nl
Tel.: ++31 +13-466.8080, Fax: .3069
http://infolab.kub.nl/people/willi/

Abstract. For integrating federated information systems, semantic interoperability is necessary to ensure that exchange of information makes sense — that the provider and requester of information have a common understanding of the 'meaning' of the requested services and data. Effective exchange of information between federated systems needs to be based on a common understanding of the transferred data. Below, we discuss the *process* of integrating federated information systems in this context:

- The process in which federated information systems are traditionally integrated in a *bottom-up* way and some problems with this approach.
- The process in which federated information systems could ideally be integrated in a *top-down* way to achieve more usable and scalable systems.
- A combined *yo-yo* approach to exploit the benefits of both strategies and to serve as a migration path form the traditional bottom-up approach towards an ideal top-down approach.

1 The Traditional Bottom-Up Process

Traditionally, the integration of federated information systems proceeds in a *bottom-up* process. Information stored in existing legacy systems is analyzed with respect to potential overlaps, whereby overlapping data in dissimilar systems describes the same or related information. The overlapping areas of related information sources are subsequently integrated. The integration is usually realized by means of mediators, federated database systems or such-like system architectures. Typical goals for the integration of existing systems are the development of global applications that access the data from multiple sources as well as consistency management of information that is stored in related systems.

Let us consider hospital information systems as an example domain where consistent data replication is one of the central problems to be solved [4, 5]. A process model for the *bottom-up* integration of departmental subsystems within hospitals is presented in [5]. As one result of such a bottom-up approach the

structure of the merged federated schema is determined by the overlaps among the local schemas, and not by the requirements of global applications. The maintenance of such integrated schemas is a problem, because those merged schemas rapidly become very complex; usually more complex than required for the actual integration goals. This situation can lead to severe scalability problems with respect to execution performance, usability and maintenance.

Figure 1 illustrates the bottom-up process for constructing the schema architecture in federated database systems which starts with an analysis of information stored in the local systems. To explain the schema types displayed in Figure 1: 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. In a first step, the local schemas are translated into component schemas in the canonical data model of the federation layer. Then, export schemas are filtered and combined into a federated schema. A *component* schema is a local schema transformed into the *canonical* data model of the federation layer. An *export* schema is derived from a component schema and defines an interface to the local data that is made available to the federation. A *federated* schema is the result of the integration of multiple export schemas, and thus provides a uniform interface for global applications. An *external* schema is a specific view on a federated schema or on a local schema which serves as a specific interface for applications (local or global). In Figure 1, no external schemas are shown. The component, export and federated schemas are defined in a so-called *canonical* data model [10]. The translation from local to component schemas eliminates the data model heterogeneity among the various local schemas. Since the component and export schemas are provided in a canonical data model, no model translation is needed on the federation layer. However, many discrepancies between the export schemas may exist. For example, similar entities may be specified at different abstraction levels, or equivalent attributes may have different data types.

As one characteristic result of bottom-up construction of the schema architecture, the structure of the common federated schema is determined by the overlaps among the local schemas. For instance, most object-oriented integration approaches resolve semantical overlappings by introducing generalized classes such that the original inheritance hierarchies are sub-hierarchies of the resulting merged hierarchy (upward inheritance principle [3,8]). These approaches take over the inheritance hierarchies from the local to the integrated schema level and adapt them to each other. Consequently, the resulting merged hierarchies become needlessly complex.

Due to the upward inheritance principle, the integrated schemas contain the sum of the classes contained in the individual component schemas. For reducing this complexity within the integrated schema, classes may be merged [7], but optimizing these merged hierarchies with respect to minimizing the number of classes may introduce new constraints; thus, complicating the integrated federated schema.

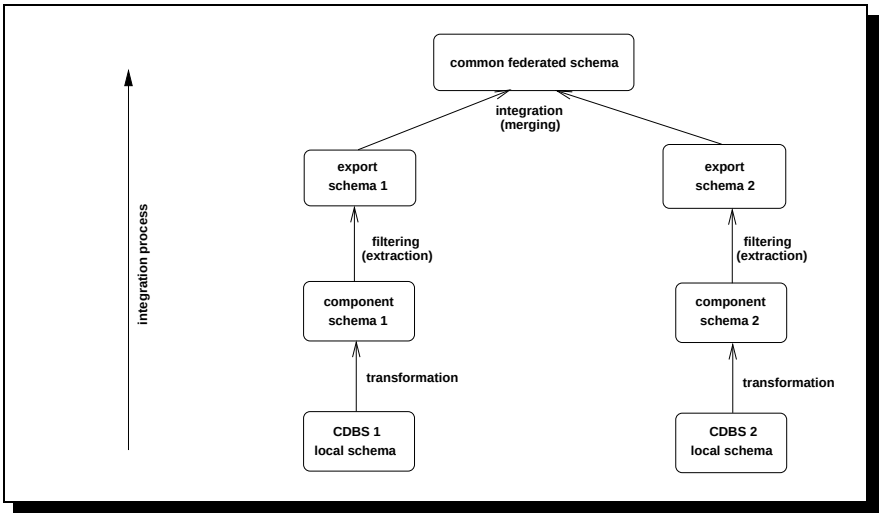

Fig. 1. Bottom-up integration on the schema level. The arrows illustrate the development process that starts with the local schemas of some existing legacy systems.

Under those traditional integration paradigms [1], the integrated view/schema depends directly on the source schemas. An integration engineer defines the desired integrated view by examining all the existing systems to be integrated. The bottom-up approach is to sum up the capacity of existing information systems in one global schema. As a result, the usability and maintainability of such integrated schemas can become a serious problem.

Another way to approach the integration of federated information systems could be a *top-down* process. Starting with a common schema that is based on the requirements of new (global) applications and on domain-specific standards, the individual local schemas are integrated into this common schema.

2 Top-Down Integration

To approach a more 'ideal' top-down integration process, let us start with a look at domain-specific software development. *Domain engineering* is an activity for building reusable components whereby the systematic creation of domain models and architectures is addressed. Domain engineering also supports *application engineering* which uses the models and architectures to build systems. The emphasis is on reuse and product lines. The Domain-Specific Software Architecture (DSSA) [11] engineering process was introduced to promote a clear distinction between domain and application requirements. A Domain-Specific Software Architecture consists of a domain model and a reference architecture as modeled in Figure 2. The DSSA process consists of domain analysis, architecture modeling, design and implementation stages as illustrated in Figure 3.

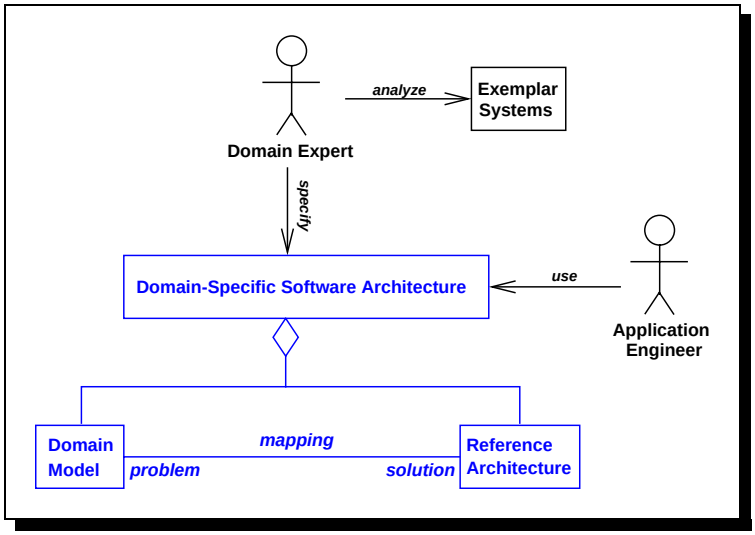

Fig. 2. Relations between some roles and artifacts in the DSSA engineering process. Hollow diamonds indicate part-of relations. We use the UML notation for actors to model the roles [2].

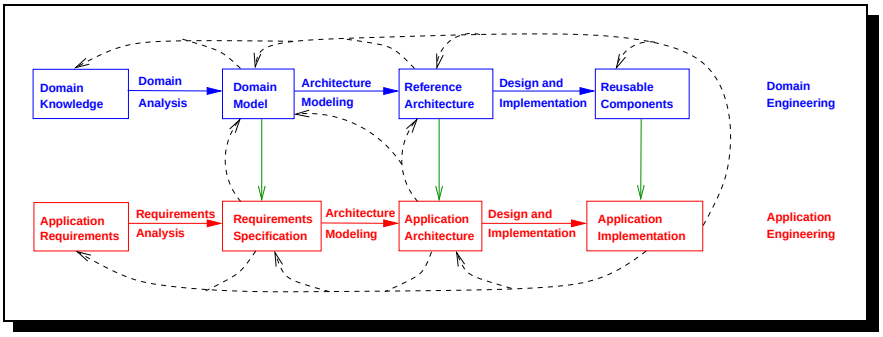

Fig. 3. The DSSA engineering process. In application engineering, software systems are developed from reusable components created by a domain engineering process. As indicated by the dashed arrows, various forms of feedback are possible.

Domain models represent the set of requirements that are common to systems within a product line. There may be many domains, or areas of expertise, represented in a single product line and a single domain may span multiple product lines. *Domain analysis* is the process of identifying, collecting, organizing, and representing the relevant information in a domain, based upon the study of existing systems and their development histories and knowledge captured from domain experts.

In *application engineering*, a developer uses the domain models within the product line to understand the capabilities offered by the reference architecture and specifies a system for development. The developer then uses the reference architecture to build the system. Architectural modeling is used to represent the framework for constructing the applications. An architectural model is developed in this phase from which detailed design and component construction can be done. The *architecture* of a software system defines that system in terms of components and interactions/connections among those components. 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 [9].

Domain engineering and application engineering are complementary, interacting, parallel processes that comprise a reuse-oriented software production system. An application engineering process should develop software systems from reusable components created by a domain engineering process (see Figure 3). The focus of application engineering is a single system whereas the focus of domain engineering is on multiple related systems within a domain. Typical application engineering activities include using a domain model to identify customer requirements and a (generic) reference architecture to specify an application architecture.

Engineering of federated information systems will usually require the integration of existing information sources. Despite this fact, we argue that the integration process should proceed in a *top-down* way starting with the data models that are common to all the involved local systems, i.e. with domain models. For such a top-down integration of those federated information systems, we propose the use of domain-specific standards (such as STEP, OAGIS, Z39.50, etc.) as the basis for the federated schemas in the schema architecture of federated database systems. Figure 4 illustrates the top-down process for constructing the schema architecture which starts with the common federated data schema that should be based on requirements of global applications and on some standards. The individual local schemas are 'hooked' into this common schema via the export and component schemas. To 'hook' a local information system/schema into such a common domain schema, the responsive integration engineer has to understand his or her local information system and the corresponding domain model, but does *not* have to understand the other component information systems/schemas.

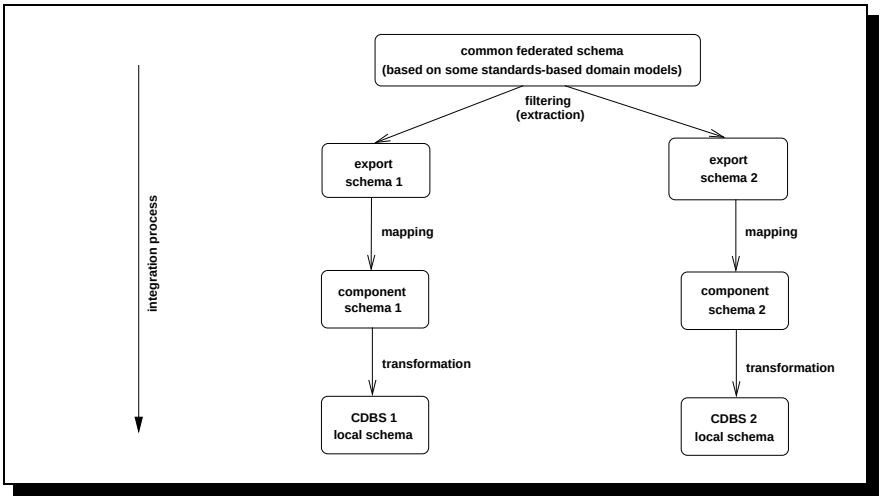

Fig. 4. Top-down integration on the schema level. The arrows illustrate the development process that starts with the common standards-based schema.

3 A Combined Yo-Yo Approach

In practice, we can also expect a *yo-yo* approach as illustrated in Figure 5: the integration process alternates with bottom-up and top-down steps. A top-down process can be expected to support global applications — such as workflow management or decision support systems. A bottom-up process can be expected in the case that some local systems regularly need information from other systems — e.g., a laboratory system that needs the particulars of patients. The bottom-up process may provide input for extending the domain models. Figure 6 illustrates the relationships between such a combined *yo-yo* approach and domain/application engineering.

4 Summary

Usually, the result of the traditional bottom-up approach to the integration of federated information systems is that the structure of the federated schemas is determined by the overlaps among the local schemas. As opposed to the bottom-up approach, with the top-down approach the structure of the common schema is *not* determined by the overlaps among the local schemas, but by domain-specific models, which become the basis for *semantic* interoperability. Particularly, the use of domain-specific standards as the basis for the common schema alleviates the integration of commercial components that offer standards-compliant interfaces. The use of such standards within the integration system also encourages a (smooth) migration towards modern standards-compliant systems.

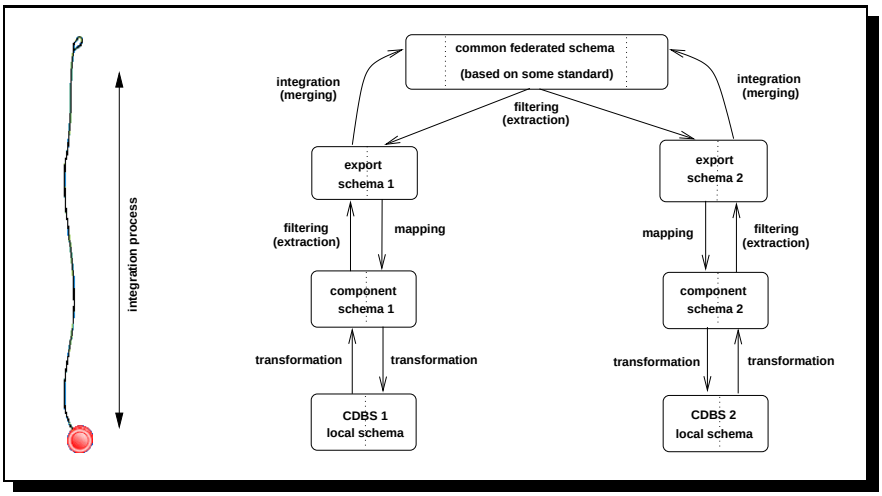

Fig. 5. A combined *yo-yo* approach to exploit the benefits of both strategies.

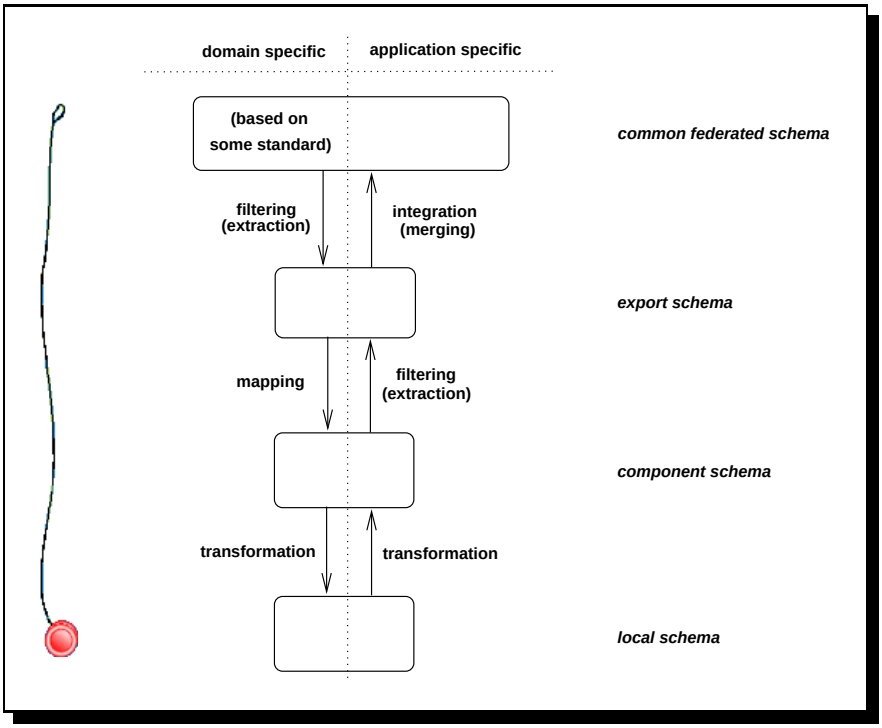

Fig. 6. The combined *yo-yo* approach related to domain and application engineering.

To quote Jim Kleewein on practical issues with commercial use of federated databases [6]:

> "Schema integration is one aspect of usability that impedes federation. There are often thousands of tables or views involved in a federation making maintenance of a global schema difficult."

Basing the global schema on standards-based domain models avoids changes to the fundamental structure of this schema, making integration more usable and scalable.

References

1. C. Batini, M. Lenzerini, and S.B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
2. G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide*. Object Technology Series. Addison-Wesley, Reading, MA, 1998.
3. U. Dayal and H.-Y. Hwang. View Definition and Generalization for Database Integration in a Multidatabase System. *IEEE Transactions on Software Engineering*, 10(6):628–644, 1984.
4. W. Hasselbring. Federated integration of replicated information within hospitals. *International Journal on Digital Libraries*, 1(3):192–208, November 1997.
5. M.N. Kamel and M. Zviran. A methodology for integrating heterogeneous data bases in a hospital environment. *Journal of Systems and Software*, 15(3):251–260, July 1991.
6. J. Kleewein. Practical issues with commercial use of federated databases. In *Proc. 22th International Conference on Very Large Data Bases (VLDB'96)*, page 580, Bombay, India, September 1996. Morgan Kaufmann.
7. I. Schmitt and G. Saake. Merging inheritance hierarchies for database integration. In M. Halper, editor, *Proc. Third IFCIS International Conference on Cooperative Information Systems (CoopIS'98)*, pages 322–331, New York City, NY, August 1998. IEEE Computer Society Press.
8. M. Schrefl and E.J. Neuhold. Object Class Definition by Generalization Using Upward Inheritance. In *Proceedings 4th International Conference on Data Engineering (ICDE'88)*, pages 4–13. IEEE Computer Society Press, 1988.
9. M. Shaw and D. Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice Hall, 1996.
10. A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.
11. R.N. Taylor, W.J. Tracz, and L. Coglianese. Software development using domain-specific software architectures. *ACM SIGSOFT Software Engineering Notes*, 20(5):27–38, December 1995.