Reference Architecture Modeling with the UML and Vital: A Comparative Study

Willi Hasselbring
Software Engineering Group, University of Oldenburg
D-26111 Oldenburg, Germany
Tel: +49 441/798-4517
Email: hasselbring@informatik.uni-oldenburg.de

ABSTRACT
Software reference architectures aim at supporting software product lines, describing the *commonalities* as well as the *variabilities* among individual products in a family. Many notations for architectural representation do exist, but only a few explicitly support modeling of software *reference* architectures.

The Variability and Dependency Model (Vital) offers such a notation which is specifically designed for describing software product line reference architectures [7]. Vital complements existing notations by focusing on issues only relevant to reference architectures and assuming that all other required architectural information is properly described by other means.

We present a comparative case study for reference architecture modeling. Exemplarily, the generic reference architecture for schemas in federated database systems is described in the UML and in Vital. The goal is to see how the UML can be employed and possibly extended for *reference* architecture modeling. The comparison of the UML with a notation that was explicitly designed for reference architecture modeling should deliver appropriate input for further development of the UML standard.

Keywords
Software Architecture, Reference Architecture, Architecture Description.

1 INTRODUCTION

Software architectures support reuse by serving as frameworks for understanding families of systems, which may be called *product lines* [4, 6, 9]. *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 [13]. A DSSA consists of a domain model and a reference architecture, and guides in reusing components for developing product lines as modeled in Figure 1. Appropriate management of component libraries is essential for successful reuse. The DSSA process consists of domain analysis, architecture modeling, design and implementation stages.

2 SOFTWARE REFERENCE ARCHITECTURES FOR PRODUCT LINES

When describing a reference architecture for a product line, it is required to cover a variety of similar systems. This is an important difference to describing the architecture of individual software systems. Anyway, a reference architecture should be described in *one* model with a generic nature. Every member of the product line represents an instantiation of this generic structure [10]. The reference architecture represents all possible architectural elements together with the rules for instantiating the individual product line members. Therefore, reference architectures describe invariant and variable elements. Dependencies between those elements may exist: there can be existence dependencies and elements may exclude each other. The reference architecture has to represent all valid family members and must not represent invalid members.

3 AN EXAMPLE PRODUCT FAMILY: FEDERATED DATABASE SYSTEMS

Federated database systems are a research topic for many years, and database systems in general are a well-understood research area with a reasonable formal underpinning. Therefore, we selected this product family as an example.

A federated database system integrates heterogeneous, autonomous database systems, whereby both local applications and global applications accessing multiple component database systems are supported [12]. Such a federated database system is a complex *system of systems* which requires a well designed organization at the software architecture level. A problem that federated database systems face, is the organization of schemas in a schema architecture.

Position paper submitted to the 1st ICSE Workshop on Describing Software Architecture with UML.

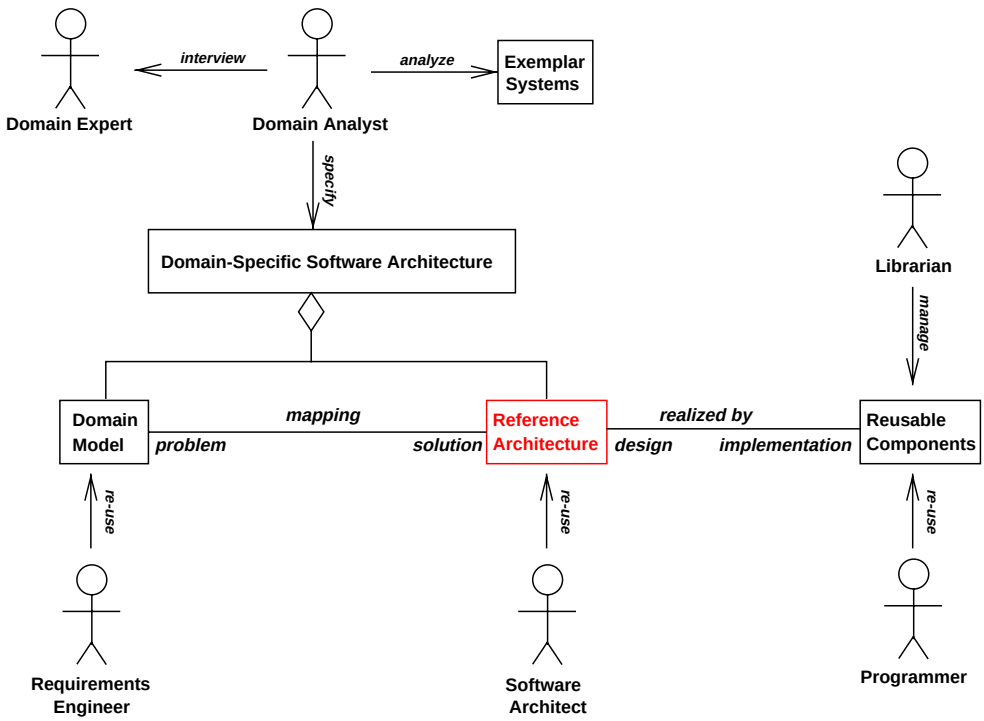

Figure 1: Relations between some roles and artifacts in the DSSA engineering process. Reference architectures for product lines are essential in this context. We use the UML notation for actors to model the roles [3].

For federated database systems, the traditional three-level database schema architecture 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 [12]. As reported in [5], this reference schema architecture is generally accepted as the basic structure in federated database systems or at least for comparison with other specific architectures.

A reference architecture for those schemas is useful to clarify the various issues and choices within complex federated systems. It provides the framework in which to understand, categorize and compare different architectural options for developing specific systems. Reference architectures are the structures used to build systems in a *product line*.

In [8], we relate the concepts of this reference architecture to those realized in some specific federated database management systems. The purpose was to show how the reference architecture can be compared to the architectures of various federated database systems. Such a representation supports the task of studying and comparing these systems. In the present paper, we only consider the generic reference architecture and compare its representation in the UML and in Vital. The goal is to see how the UML can be employed and possibly extended for reference architecture modeling. The comparison with a notation that was explicitly designed for reference architecture modeling is expected to deliver input for this task.

4 THE UML SPECIFICATION

Figure 2 displays the reference schema architecture for federated database systems described in the UML notation for class diagrams [3]. In this model, some of the constraints and options for the architecture, which are informally discussed in [12], are defined by means of the *multiplicities* at the associations and other notational means.

The different schema types in Figure 2 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.

Component Schema: A Component Schema is a Local Schema transformed into the (canonical) data model of the federation layer.

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.

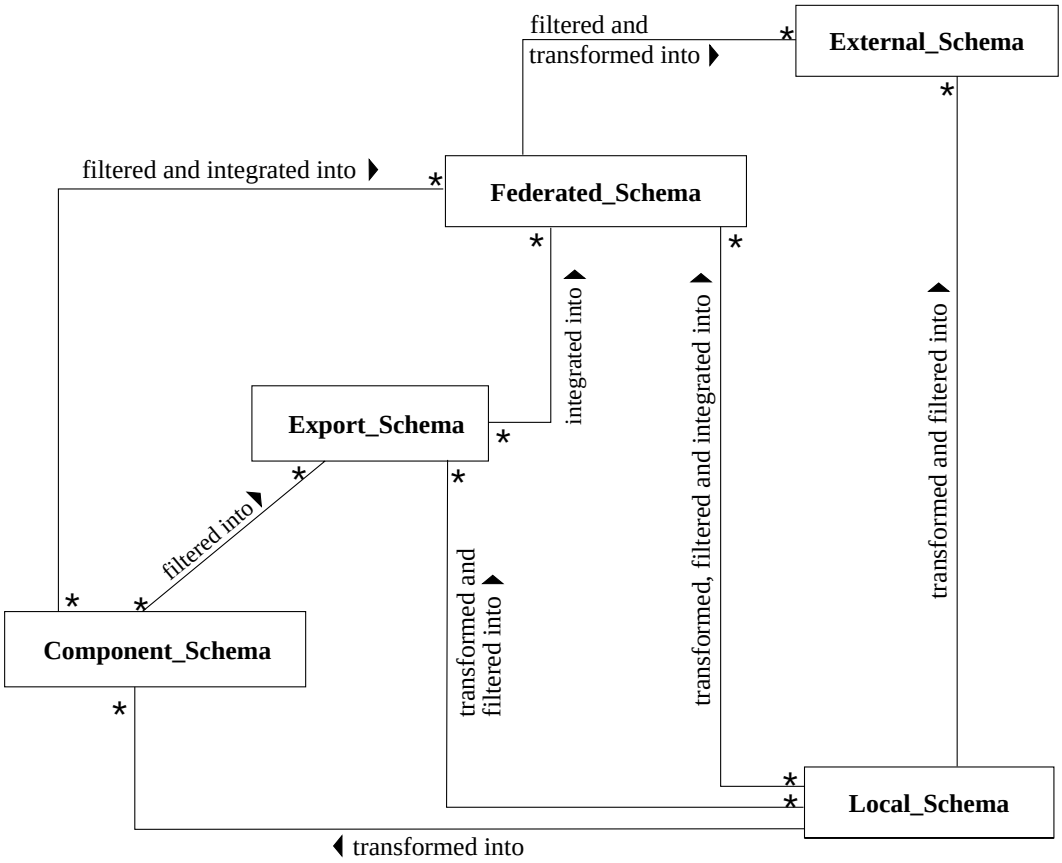

Figure 2: Extract of the UML class diagram for the 5-level reference schema architecture for federated database systems.

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 from the canonical data model. Basically, External Schemas serve as specific interfaces for applications (local or global).

This schema architecture, which is managed by the federated database management system, specifies the dependencies/correspondences among the individual schemas. 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 concurs with the 5-level schema architecture of [12]. 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 External Schemas.

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

- 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.

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 [3].

5 THE VARIABILITY AND DEPENDENCY MODEL VITAL

Vital, focuses on the variabilities of product lines [7]. It is meant to complement other architectural views reflecting other essential architectural information, such as could be provided by the UML.

Vital offers both a textual description language and a graphical representation. Both have the same expressiveness. Here, only the graphical representation is employed which offers the following notational elements (the explanation is adopted from [7]):

- Rectangles represent elements of the software system.
  - A white rectangle represents a mandatory element. Mandatory elements are prescribed within their scope of definition. On the top-level they constitute the commonality of a product family.
  - A shaded rectangle stands for an optional architectural element. It manifests a variable part within a software system.
  - If further refinement is concealed within an element then this is marked through a triangle at the bottom right corner. A solid triangle identifies an object which contains mandatory elements only.
  - An element with a dotted triangle keeps at least one optional subelement inside. Hence it holds variability.

  Those rectangles are very similar to classes in the UML. The different types of rectangles could be modeled via stereotypes.
- Ellipses represent artificial elements used for structuring the system's architecture. Hence they do not have corresponding components within the software system.
  - A white ellipse represents a mandatory architectural part. As a mandatory element it is prescribed within its scope of definition.
  - A shaded ellipse stands for an optional architectural structure. It comprises variable parts of a software system.
  - Fine grain structures can be hidden from the eye of the beholder. Virtual elements are then marked on the right tip. A solid tip defines a virtual object which contains mandatory elements only.
  - A structure with a dotted tip keeps at least one optional subelement inside (i.e., it has been applied to subsume variable parts).

  The UML does not directly support a concept similar to what these ellipses are intended for. Packages are somewhat similar, but not identical.
- Lines represent links between two relation points. A relation consists of two relation points and a link connecting both. Multiple solid lines starting from an identical relation point are connected with a logical AND to specify inclusive dependency which implies that the inclusion of one element requires the inclusion of further elements. Multiple dashed lines starting from an identical relation point are connected with a logical XOR to specify exclusive dependency which expresses incoherences between elements.

  In the UML, these lines can be modeled as association links, whereby the different kinds of lines could be specified via stereotypes or constraints.
- Circle and rhombus represent two different kinds of relation points. Elements can only be connected via relation points. The meaning of the relation is reflected by the appearance of the relation point.
  - A white circle is a relation point that signals no dependency. It is applied to visualize the completeness of the relation. Through keeping the representation consistent it facilitates the identification of incorrect connections.
  - A solid circle represents a prerequisite relation. All incoming links stem from architectural parts depending on this element. This is used to support inclusive dependencies.
  - Architectural parts connected to a shaded circle are affected by the corresponding element. The appearance (existence) of the element has an influence on their own constitution. Thus a shaded circle represents the visualization of relations between variable subelements at a higher hierarchical level.
  - A solid rhombus represents the exclusive dependencies. Only links from other rhombi may be connected.

  In the UML, these symbols would be modeled as different types of association links (e.g., via stereotypes). Vital does not offer multiplicities as they are offered in the UML for association links.
- Constraints are used to express complex dependencies between architectural elements, very similar to constraints (OCL) in the UML.

Due to space limitation we cannot present the Vital reference model for federated database systems in this short position paper. We can only discuss general commonalities and differences. In an extended version, the Vital model and a UML model — revised on the basis of the Vital model — shall be presented to illustrate more details of the conclusions on what the UML should offer for reference architecture modeling.

6 SUMMARY

We model the generic reference architecture for schemas in federated database systems and compare its representation in the UML and in Vital. The goal is to see how the UML can be employed and possibly extended for reference architecture modeling. The comparison with a notation that was explicitly designed for reference architecture modeling shall deliver input for this task.

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 [1, 2, 11]. 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. An architecture embodies decisions about quality properties. It represents the earliest opportunity for evaluating those decisions. Furthermore, reusability of components depends on how strongly coupled they are with other components in the system architecture. Performance depends largely upon the complexity of the necessary coordination, in particular when the components are physically distributed processes. For product lines reference architectures are required.

Such reference architectures need to emphasize on two kinds of dependency relations that can exist between elements: inclusive and exclusive dependencies.

- Inclusive dependency implies that the inclusion of an optional element requires the inclusion of further elements.
- Exclusive dependency expresses incoherences between elements. Thus one variable element may exclude others from being integrated within the same application.

Consequently, architectural description notations supporting reference architectures must be able to express the variability and dependency constraints described above. If the UML is meant to describe software reference architectures, which aim at supporting software product families, explicit notations reflecting the *commonalities* as well as the *variabilities* among individual products in a family should to offered to the software architect. The forthcoming UML 2.0 should explicitly support the software architect with modeling reference architectures for product lines.

REFERENCES

[1] BARROCA, L., HALL, J., AND HALL, P., Eds. *Software Architectures: Advances and Applications*. Springer-Verlag, London, 2000.

[2] BASS, L., CLEMENTS, P., AND KAZMAN, R. *Software Architecture in Practice*. Addison-Wesley, Reading, MA, 1998.

[3] BOOCH, G., RUMBAUGH, J., AND JACOBSON, I. *Unified Modeling Language User Guide*. Object Technology Series. Addison-Wesley, Reading, MA, 1999.

[4] BOSCH, J. *Design & Use of Software Architectures: Adopting and evolving a product-line approach*. Addison-Wesley, Harlow, England, 2000.

[5] CONRAD, S., EAGLESTONE, B., HASSELBRING, W., ROANTREE, M., SALTOR, F., SCHÖNHOFF, M., STRÄSSLER, M., AND VERMEER, M. Research Issues in Federated Database Systems (Report of EFDBS '97 Workshop). *SIGMOD Record 26*, 4 (Dec. 1997), 54–56.

[6] DIKEL, D., KANE, D., ORNBURN, S., LOFTUS, W., AND WILSON, J. Applying software product-line architecture. *Commun. ACM 30*, 8 (Aug. 1997), 49–55.

[7] GACEK, C., AND VUKOVIC, A. Vital: Representing software reference architectures. In *Proc. Fourth International Software Architecture Workshop ISAW-4* (Limerick, Ireland, June 2000), pp. 105–109.

[8] HASSELBRING, W. Formalizing and comparing software architectures of federated database management systems. In *Proc. Fourth International Software Architecture Workshop ISAW-4* (Limerick, Ireland, June 2000), pp. 126–130.

[9] MACALA, R., STUCKEY, L., AND GROSS, D. Managing domain-specific, product-line development. *IEEE Software* (May 1996), 57–66.

[10] PERRY, D. Generic architecture descriptions for product lines. In *Proc. Second International Workshop on Development and Evolution of Software Architectures for Product Families* (1998), no. 1429 in Lecture Notes in Computer Science, Springer Verlag, pp. 51–56.

[11] SHAW, M., AND GARLAN, D. *Software architecture: perspectives on an emerging discipline*. Prentice Hall, 1996.

[12] SHETH, A., AND LARSON, J. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys 22*, 3 (1990), 183–236.

[13] TAYLOR, R., TRACZ, W., AND COGLIANESE, L. Software development using domain-specific software architectures. *ACM SIGSOFT Software Engineering Notes 20*, 5 (Dec. 1995), 27–38.