

A Meta-Model based Approach for the Combination of Models in multiple Languages

Dr. Eckhardt Holz
holz@informatik.hu-berlin.de
Humboldt University Berlin, Dept. of Computer Science
Rudower Chaussee 5,
D-10099 Berlin Germany

ABSTRACT

The provision of efficient methods and tool support for the development and engineering of distributed systems is a key enabling factor for the evolution of Information Technology. Distributed as well as telecommunication systems consist of components which are distributed across networks and have to cope with concurrency, autonomy, synchronization, and communication issues. The development of correct and efficient distributed systems is a complex and complicated task. CASE tools support the design and partially the validation of distributed systems in the modelling phase.

KEYWORDS

Languages, Object Oriented Implementation, Model Combination, Meta-Modelling, UML, SDL

1. Introduction

A first code generation step from object oriented design models leads in many cases to functional prototypes. Within the telecommunication domain traditionally formal specification and description techniques like SDL (ITU-T Specification and Description Language) or MSC (ITU-T Message Sequence Charts) play an important role for the specification and development of such design models. Original application areas of these languages are

- requirements and system specifications,
- standards and recommendations (by standardization bodies like ITU-T and ISO),
- system design and implementation specifications and
- system test specifications

where the term system usually comprises telecommunication protocols and services.

However, these techniques are not only used for the

notation of specifications standards but are also directly applied in the design process. Tools support this with a variety of features, from graphical editors over verification & validation tools and analyzers to code generators.

Main purpose of the analysis tools is to improve the reliability of the specifications and of the generated implementations. Techniques applied here range from mathematical proving and reachability analysis to simulation and testing. Especially the latter two are widely supported by tools for real-life applications, whereas reachability analysis as well as proving technologies are due to the complexity limited to smaller applications resp. partial system specifications.

In order to extend the applicability of SDL to other domains, rules for a combination of SDL and UML have been developed and are now standardized by the ITU. Unfortunately, the simulation or prototype evaluation of such combined system models using the dedicated tools is limited:

- SDL tools do not accept UML models and
- UML tools do not accept SDL models.

2. Foundation

The combination method is based on a set of well defined concepts, that form the concept space or meta-model for the method. Mappings are defined, that relate a concept or a set of concepts to elements of a concrete target language (i.e UML or SDL).

Adhering to this idea, the combination comprises three parts, which are:

- the meta-model, that defines a language- and tool-independent core terminology for the specification and design of distributed applications,
- a set of rules relating the core concept space with the concept spaces of the actual languages, and

- a single or multiple sets of mapping rules, which enable a smooth transition from the design to the implementation by concrete tools.

2.1. Meta-Modelling

The extensive application of models and modelling techniques within the scope of the development and the use of CORBA-systems lead to the requirement for a unique and standardized framework for the management, manipulation and exchange of models and meta-models. This need has been addressed by the OMG with the Meta-Object-Facility (MOF, [1]). The architecture of MOF is based on the traditional four-layer approach to meta-modelling:

- Layer M0 - the instances: information (data) that describe a concrete system at a fixed point in time. This layer consists of instances of elements of the M1-layer.
- Layer M1- the model: definition of the structure and behavior of a system using a well defined set of general concepts. An M1-model consists of M2-layer instances.
- Layer M2 - the meta-model: The definition of the elements and the structure of a concept space (i.e. the modelling language). An M2-layer model consists of instances of the M3-layer.
- Layer M3 - the meta-meta-model: The definition of the elements and the structure for the description of a meta-model.

Although further levels (i.e. for the definition of the elements to specify the M3-layer) seem to be necessary, MOF as well as other meta-modelling frameworks stop at M3. This is due to the application of a so-called hard-wired meta-meta-model: Elements and structure of the M3-layer are directly derived from a well-known formalism, in case of MOF object-orientation. In fact, the MOF concepts are defined using MOF itself. The MOF-model consists of the following concepts for the definition of meta-models:

- **Classes:** Classes are first-class modelling constructs. Instances of classes (at M1-layer) have identity, state and behavior. The structural features of classes are attributes, operations and references. Classes can be organized in a specialization/generalization hierarchy.
- **Associations:** Associations reflect binary relationships between classes. Instances of associations at the M1-layer are links between class instances and do neither have state nor identity. Properties of association ends may be used to specify the name, the multiplicity or the type of the association end. MOF distinguishes between aggregate (composite) and non-aggregate associations.

- **Data types:** Data types are used to specify types whose values have no identity. Currently MOF comprises the CORBA data types and IDL interface types.
- **Packages:** The purpose of packages is to organize (modularize, partition and package) meta-models. Applicable mechanisms here are generalization, nesting, import and clustering.

The MOF specification defines these concepts as well as other, supporting concepts in detail. The interface to the conceptual framework of MOF is formally defined in the CORBA-IDL modules **Model** (meta-model specific interfaces) and **Reflective** (generic interfaces). All interfaces in **Model** directly or indirectly inherit from **Reflective** interfaces. The MOF interfaces allow to

- stepwise create a new meta-model in the MOF by creating new objects,
- change an existing meta-model in the MOF,
- extract information from a meta-model using query- and traversal functions, and
- request a validation of the meta-model.

In addition to that, functions are defined to produce an external representation of a meta-model (externalize) or to create a meta-model in MOF from such an external representation (internalize).

The advantage of the MOF-approach is to make the definition of (meta)-models independent of the concrete application domain of the models and to provide a concise and unique set of concepts for the definition of meta-models. Moreover, multiple meta-models can be managed by the MOF. Further investigations have to show, how relations between meta-models can be utilized as a basis for a transformation of models based on these meta-models.

3. Common Concept Core

Starting from the informal definition of the common concept space, we have applied the MOF to specify a meta-model that captures the concepts of our concept space.

The meta-model consists of a set of packages. According to the current application domain of the concept space, we chose to base all definitions on the meta-model for UML as defined in [2]. The complete meta-model is stored in the Meta Object Facility.

In order to determine the set of concepts that forms the common concept core, it has to be decided which form of model combination shall be supported. In [3] three general types of model combination within the software engineering process are identified:

- *Process Driven Combination:* Models of one design phase or activity serve as starting point for the next phase or activity and different modelling languages

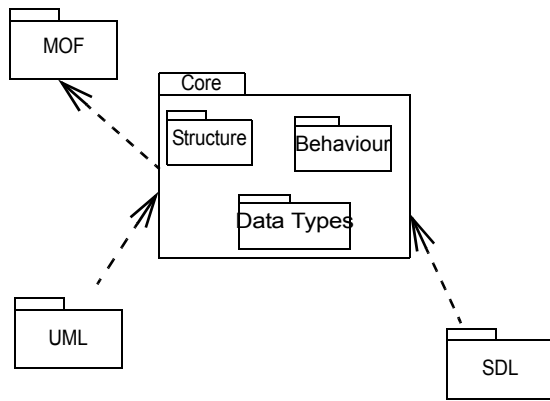


Figure 1. Concept Space Structure

are used for the activities or phases. The models are chronologically ordered, i.e. they are not used at the same time.

- *Abstraction Driven Combination*: The single models describe the system within the same design phase but using different abstractions (views) and these views deploy different languages. In this case an overall model can be obtained by a superimposition of the sub-models.
- *Structural or Architectural Combination*: The single models describe different components or sub-systems of the overall system and different domain or task specific languages are applied for the sub-models. The overall model can be obtained by a composition/aggregation of the sub-models.

The first two approaches serve a smooth transition between the different phases in the software design and a dedicated capture of design decisions. If code generation or prototype production is the goal the emphasis lies on structural combination and abstraction driven combination.

The structural combination requires the modelling languages to be able to:

- specify an overall model as a composition of interconnected subsystem and component models,
- specify interfaces between the components and subsystems, and
- specify the components and subsystems as open models (i.e. they must be able to communicate with their environment).

Due to the independence of the single submodels structural combination enables the autonomous verification, validation and evolution of the submodels. However, the interfaces specified for the interconnection must not be changed.

The application of the structural combination requires at least a set of concepts to specify architectural features, which comprise according to Bachman [4] and Burbach [5] concepts to identify:

- subsystems and components
- connectors (connections, protocols), and
- connection points (ports and interfaces).

While dedicated architecture specification languages directly contain such language constructs (cf. ACME - Garlan et.al, [6]), general software design languages like SDL or UML require additional concepts and rules.

The following table lists the common concept space for the structural combination of SDL and UML models and gives for each concept a short informal explanation:

Concept name	Semantics
ExternalUnit	Declares the existence of another submodel within the current model. Connections between the current submodel and the ExternalUnit are declared in the current model.
Port	Potential endpoints for connections between submodels. Ports are elements of an ExternalUnit. Each port has a direction indication (in, out, in/out) and a set of communication capabilities.
Interface	Specifies a communication capability in terms of operation signatures, signals and attributes. If an interface is assigned to an in-port of an ExternalUnit, the unit provides the operations of the interface to clients, if the interface is assigned to an in-port, the External requires the operations.
Class	Specifies parameters and return values of operations and signals.

Table 1 Informal concept definitions

These concepts are formally defined by a meta-model and stored in a MOF-compliant repository. An extract of the meta-model is shown in Figure 2. Using the generation and mapping features a dedicated repository for the concept space as well as an XML-based exchange format can be generated.

4. Combined Analysis

The combined analysis has to be considered at two levels - the language level and the simulator level (cf. Figure 3.).

The language level comprises the SDL specification part and the UML specification part both with additional definitions or annotations. These additional elements express in a very abstract way the existence of the UML (SDL) part in the SDL (UML) specification in form of a bidirectional interface description based on the common concept core. At the language level the information

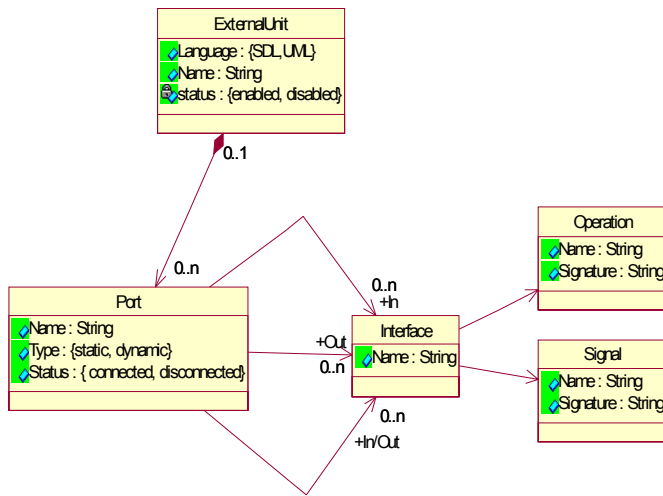


Figure 2. Concept Core - Elements

exchange between both parts is described abstractly by using methods of the interface.

The simulator or prototype level describes the implementation concepts, i.e. the connection between the UML operation calls and signals defined in the interface of the SDL part and the SDL procedure calls and signals defined in the interface of the UML part as well as the realization of the synchronization of the tools.

Introducing interfaces to the respective other part at the language level allows the separate validation and verification of the SDL and UML part. As long as the interface between both parts is not modified, the separate development and refinement of one part does not influence the other part. Moreover, the resulting specifications do not

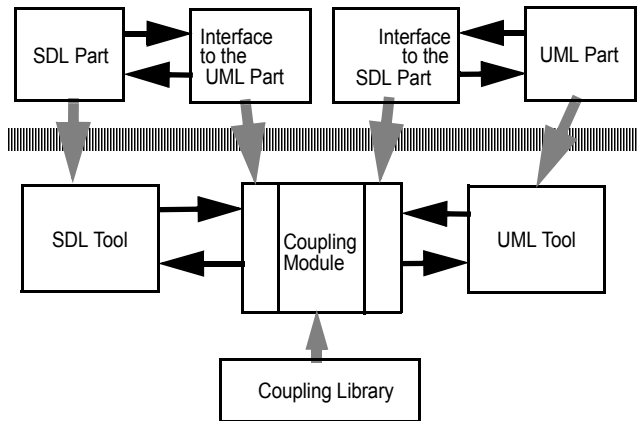


Figure 3. Coupling Architecture

depend on the realization of the simulation or prototype execution.

4.1. The Language Level

The following paragraphs give an overview of the approach

to reflect the common concepts in the two languages used for the hybrid specification. Due to the different characteristics of the two languages the strategies applied for them vary. Whereas UML includes extensibility features to adapt, specialize or extend the language according to the needs of the application domain, similar features are not available in SDL.

Concept Core Reflection in UML

The elements of the common concept core are introduced as specializations of standard UML meta-model elements in form of stereotypes and organized in the package *ExternalSDL*. Table 2 lists the stereotypes reflecting the core elements depicted in Fig. 2. The concept *Port* has

Stereotype Name	Specialized UML Concept
«ExternalUnit»	Component
«SDLInterface»	Interface
«SDLOperation»	Operation
«SDLSignal»	Operation

Table 2 Concept Reflection in UML

currently no visual reflection in UML. Each interface is assigned to an own unique implicit port. In an actual UML model an SDL subsystem is reflected by an UML component with the stereotype «ExternalUnit» attached to it. Provided and required Interfaces are specified by appropriate interfaces. In Fig. 4 the UML model refers to an

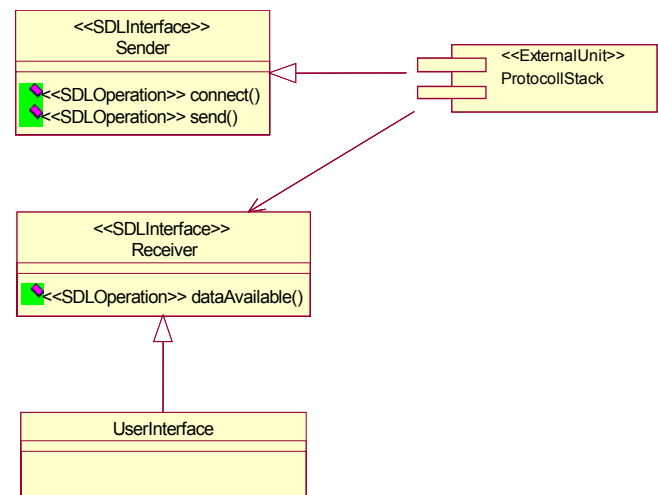


Figure 4. Interface to an SDL part (example)

SDL part *ProtocollStack*, which provides the interface *Sender* and requires the interface *Receiver*. The UML class *UserInterface* implements the required interface.

Concept Core Reflection in SDL

In SDL the common elements are defined as abstract model elements which are again organized in an package (named *ExternalUML*). Table 3 contains an excerpt of that package with the definitions for the elements in Fig. 2.

Common Core Element SDL Definition	
ExternalUnit	abstract block type ExternalUnit
Interface	interface UMLInterface
Port	(reflected by gates of block type External Unit)
Operation	(reflected by procedure of UML-Interface)
Signal	(reflected by signals of UML-Interface)

Table 3 Concept Reflection in SDL

In an SDL specification of an application the UML part is specified as a specialization of the abstract SDL model elements as shown in Fig. 5. Here we see the UML class *UserInterface* as the ExternalUnit, it provides the interface *Receiver* and requires the interface *Sender*. Additionally it is shown, that both interfaces can be accessed through the same port (SDL gate). The SDL blocks *SenderEngine* and *ReceiverEngine* are connected to an instance of *UserInterface* via the port.

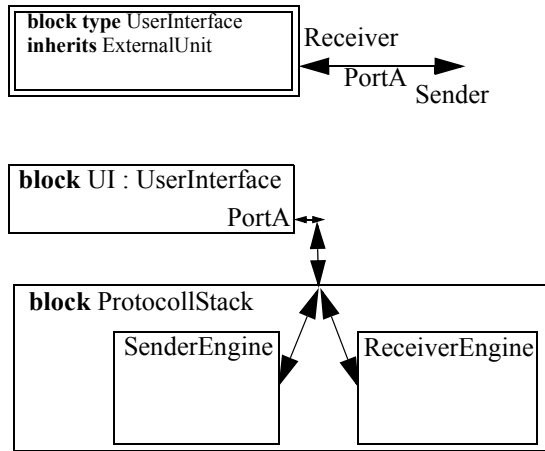


Figure 5. Interface to an UML part (example)

The modelling tools are connected to the repository for the common concept space. This enables the following features:

- store the description of an external interface in the repository (as an instance),
- lookup a description in the repository,

- change such an entry,
- delete an entry.

Therefore the definition of the interface between the two models has to be given only in one language, the complementary definition in the respective other language can be imported from the repository.

4.2. Prototype Level

Two topics have to be considered at the prototype level:

- the access to the other submodels from within the single prototypes,
- the connection between the prototypes (simulators).

The overall principle of the connection is an middleware based approach (e.g. based on CORBA). The first topic has to be solved by the code generators that produce programming language code out of the single submodels (UML-code generator, SDL-code generator). For each interface within the repository stubs and skeletons are produced according to the language binding of the middleware in use. These elements are used in the prototypes to call procedures to or accept procedure calls from the respective other prototypes.

Additional methods are introduced for the initialization of the overall system and the discovery of the single prototypes to form the overall simulators. These methods are again based on the common repository: Each prototype that provides an interface registers itself in the repository with a new entry of the appropriate interface type and its own identity (remote object reference), a client who requires an interface performs a look-up operation on the repository to obtain the reference to the provider.

A special coupling module realizes the cooperation between the SDL prototype and the UML prototype of the system. For that reason it contains two interfaces: one to the SDL part, and one to the UML part. The SDL interface is an C++ structure which represents an SDL block; the UML interface is a C++ structure which represents a UML

component (cf. Fig. 7).

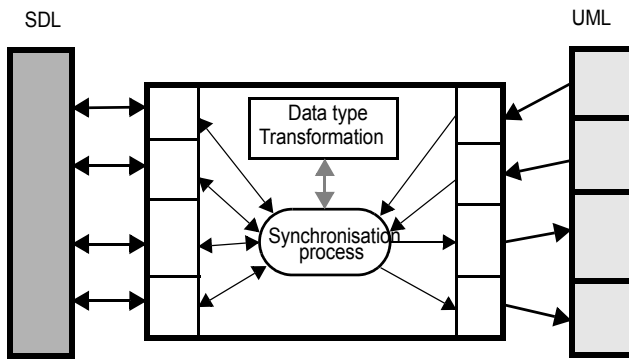


Figure 6. The Architecture of the Coupling Module

The Coupling Module is responsible for:

- the connection between SDL and UML part,
- the marshalling/unmarshalling of parameter values,
- the exchange of information between SDL and UML part (including the transformation of data types), and
- the synchronization between SDL and UML part.

The transformations are based on a limited set of basic data types and a set of predefined functions to map values of these data types. The mappings for more complex data types requires additional user defined transformation functions, which can be embedded in the coupling module.

5. Conclusion

The paper presented an approach to connect models on base of a meta-model approach. The SDL part and the UML part of the overall system model are developed independently of each other. Relations and connections between both parts are reflected by links to a common repository of reference points. Those reference points are directly based on a set of concepts common to both languages and specify the interface between the subsystems. Independent prototypes for the functional evaluation are then derived from the two models. The links to the reference points are mapped onto generic access functions to the common repository. A communication between both parts is now a three-step procedure:

1. Lookup-operation with the name of the reference point in the repository
2. Evaluation of the communication mode (synchronous/asynchronous)
3. Conveyance of the information.

The proposed approach for the meta-model based combination of functional prototypes enables the application of standard tools for the simulation of mixed SDL-UML-system models with no or limited modifications to the tools.

References

- [1] *Meta Object Facility (MOF) Specification V.1.3*. OMG, Framingham/USA, 2000
- [2] *OMG Unified Modeling Language Specification V.1.3*. OMG, 1999
- [3] Eckhardt Holz, *Kombination von Modellen beim Entwurf verteilter und kooperierender Systeme*, Humboldt-Universität Berlin, Germany, (to appear)
- [4] F. Bachman, L. Bass et.al., *Software Architecture Documentation in Practice: Documenting Architectural Layers*. Special Report CMU/SEI-2000-SR-004, Carnegie Mellon University, Pittsburgh/USA
- [5] R. Burbach, *A Distributed Architecture Definition Language: a DADL*, <http://www-db.stanford.edu/~burbach/dadl/>, Stanford University, 1998
- [6] D. Garlan, R. Monroe, D. Wile, *Acme: An Architecture Description Interchange Language*. Proceedings of CASCON'97, 1997.
- [7] E. Holz, M. Wasowski, D. Witaszek, *Design of Hybrid Hardware/Software Systems with INSYDE*. IEE Colloquium "Partitioning in Hardware-Software Code-signs" Proceedings, London/GB, 1995
- [8] M. Wasowski, D. Witaszek, J. Fischer, E. Holz, S. Lau, O. Kath, *Co-Simulation of Hybrid SDL and VHDL Specifications*. Proc. of 9th ESM, Prag, 1995
- [9] K. Verschaeve, *UML - SDL Round-Trip Engineering through Incremental Translation of Changes*. PhD Thesis, Vrije Universiteit Brussel, Belgien, 2001