Development of Modelling Frameworks and Viewpoints with Kitalpha

Benoît Langlois
Thales Global Services
Meudon-La-Forêt, France
benoit.langlois@thalesgroup.com

Daniel Exertier
Thales Global Services
Meudon-La-Forêt, France
daniel.exertier@thalesgroup.com

Boubekeur Zendagui
Obeo
Gif-Sur-Yvette, France
boubekeur.zendagui@obeo.fr

Abstract

A common need in system, software, and hardware engineering is to describe system architectures, especially in demanding domains such as aeronautics, defence or telecommunications. Kitalpha is an environment to develop and execute MBE (Model-Based Engineering) workbenches for description of system architecture. Kitalpha uses the DSL technique in order to develop such development environments accurately, quickly, and safely. This paper presents the main features of Kitalpha and lessons learned from a DSL-based development.

Categories and Subject Descriptors

D2.2 [**Design Tools and Techniques**]: Computer-aided software engineering (CASE); D2.6 [**Programming Environments**]: Textual environments; D.3.2 [**Programming Languages**]: Language Classifications – Very high-level languages.

General Terms: Design, Languages.

Keywords: Architecture Description; Architecture Framework; DSL; Eclipse; Kitalpha; PolarSys; Viewpoint

1. Introduction

In system, software, hardware engineering, a common need during the phases of analysis and design is to describe the architecture of a system. Several standards have been established to define a shared notation, such as the UML [10], ISO/IEC 42010 standards [6], NAF [9] or DoDAF [1]. Different categories of tools implement those standards: general purpose tools such as UML tools like Papyrus [11] which address multiple architecture standards, specialized tools which address a reduced set of standards, and DSL [7][12] (Domain-Specific Language)-based tools specialized for architecture description. Kitalpha is a tool which belongs to this last category.

Kitalpha is an Eclipse modelling project of the PolarSys [13] Working Group [2]. It is dedicated to implement modelling frameworks and viewpoints, and this in coherence with the ISO/IEC 42010 standard for description of system architecture. Kitalpha provides both a development and runtime environment to create and execute rich MBE workbenches (e.g., edition with diagrams, documentation, import/export, model transformation / analysis / validation) for system / software / hardware architects and engineers in small- to large-scale projects.

Kitalpha was initiated by Thales to develop and enrich Capella [4], a tool for system engineering. But Kitalpha is generic enough to implement different architecture framework standards (e.g., TOGAF/MODAF), proprietary method or domain-specific workbenches in the MBSE (model-based systems engineering) context.

Kitalpha is a foundation tool. Indeed, above the core architecture description of a system, the purpose is to describe engineering specialities (e.g., the non-functional concerns of safety and performance), up to architecture evaluation to facilitate the decision process of architecture alternatives for complex systems in domains such aeronautics, communication, or transportation.

This paper is structured as follows. Section 2 presents Kitalpha in the context of the ISO/IEC 42010 standard. Section 3 focuses on the DSL solution adopted by Kitalpha. Section 4 provides a set of lessons learned with Kitalpha about DSL. Section 5 concludes.

2. An environment to develop and execute MBE workbenches

Kitalpha is an environment to develop and execute MBE workbenches to describe system architecture. Kitalpha is based on the ISO/IEC 42010 standard. In this standard, an architecture framework is composed of viewpoints. Each viewpoint describes at least one system concern, such as non-functional concerns (e.g., performance, safety, security, cost), for involved stakeholders (e.g., safety engineer).

Conforming to that standard, an MBE workbench is an architecture framework which aggregates viewpoints. For its implementation, Kitalpha extends the definition of viewpoint to also consider it as an engineering extension which comes with its own

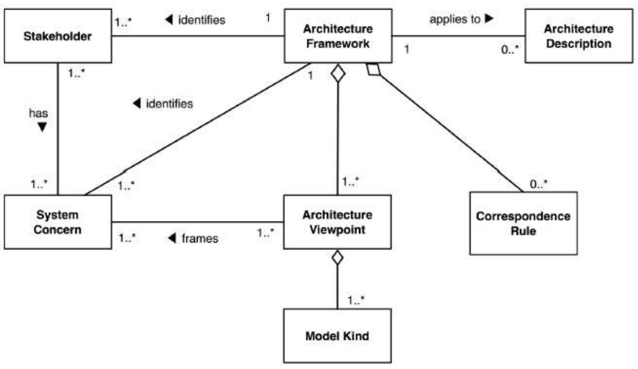


Figure 1. Architecture description based on architecture and viewpoints (ISO/IEC 42010)

metamodels, representations (e.g., diagrams, tables, user interfaces), rules (e.g., validation, analysis, transformation), services and tools to address an engineering specialty. Consequently, an MBE workbench is the result of a flexible assembly of core viewpoints extended by new ones which are, in the context of co-engineering, appropriate and valuable for specialty engineers. The set of all the viewpoints provide a solution for the complete description of a system.

Figure 2 depicts an MBE Workbench composed of core viewpoints which define the common language, representations and services to describe a system. In co-engineering, Performance, Safety, and Cost viewpoints extend here this common set. When viewpoint algorithms are too complex, computations are delegated to an external tool. A bridge enables a bidirectional exchange of viewpoint data. The complete description of a system is based on the union of all the viewpoints (i.e., the core and co-engineering viewpoints). At the workbench level, bridges ensure external communications with other MBE Workbenches or formalisms, such as UML or other DSLs.

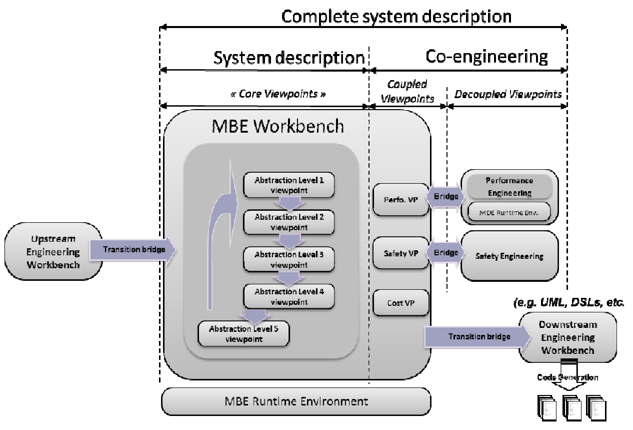


Figure 2. A MBE Workbench for architecture description

To develop MBE workbenches, a lesson learned at Thales is that designers must be autonomous in creating and maintaining their own viewpoints, without coding. Developers can enrich them afterward, for instance to implement algorithms. To meet this requirement, Kitalpha offers a development environment made of DSLs to assist designers and developers in their development activities of architecture frameworks and viewpoints. For instance, textual editors enable to declare viewpoint metamodels, user interfaces, diagrams, or services. From those DSLs, generators build all the architecture framework and viewpoint artefacts. For instance, the declaration of diagrams with an appropriate DSL is translated in Sirius [14] diagrams. Figure 3 depicts the two parallel processes of architecture framework and viewpoint development decomposed in the activities of edition with DSLs, generation and packaging to create and extend an MBE Workbench.

Kitalpha provides both development and runtime services to define, use and manage architecture frameworks and viewpoints.

The main services at development time are:

- For Architecture Framework (AF): i) definition of an AF by DSL, ii) generation of AF artifacts, iii) packaging of AF artifacts with the viewpoints it aggregates.
- For Viewpoint: i) definition of a viewpoint by DSL, ii) generation of viewpoint artifacts, iii) packaging of viewpoint artefacts.

The main services at runtime are:

- Core services: i) system architecture description with an architecture framework and its viewpoints, ii) viewpoint management in order to monitor viewpoints, iii) activation / deactivation of a viewpoint, iv) detachment / attachment of viewpoint data, v), migration of a viewpoint.
- Additional services, out of the scope of Kitalpha: versioning, collaborative work, reporting, architecture assessment, testing, simulation.

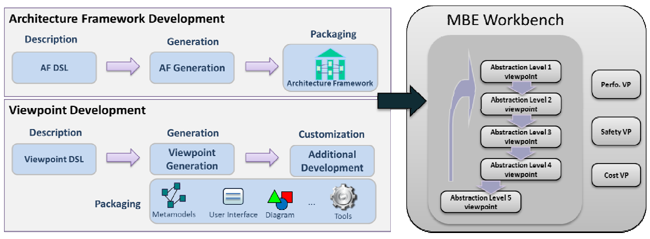
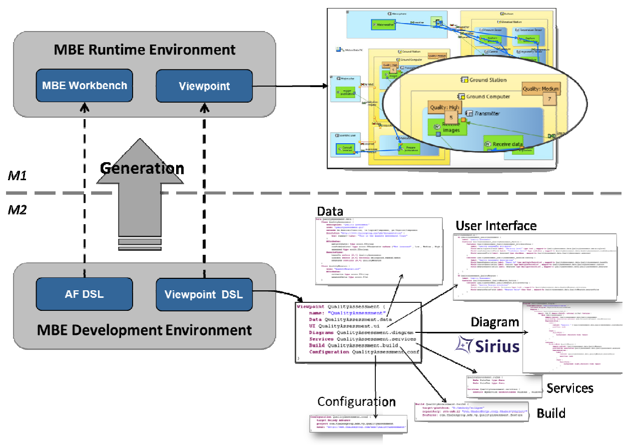


Figure 3. DSL-based development of MBE Workbenches

3. DSL Structure in Kitalpha

The two Kitalpha DSLs, for the definition of architecture framework and viewpoints of an MBE Workbench, follow the same structure. An abstract syntax defines the languages to describe architecture frameworks and viewpoints. A concrete syntax enables the designers and developers to describe architectures frameworks and viewpoints. At this stage, only a textual syntax with Xtext [16] is supported, even if the foundations are able to accept other kinds of representations (e.g., graphical or tabular). A mechanism of synchronisation translates concrete syntax into abstract syntax and vice versa.



```
Viewpoint QualityAssessment {
    name: "QualityAssessment"
    Data QualityAssessment.data
    UI QualityAssessment.ui
    Diagrams QualityAssessment.diagram
    Services QualityAssessment.services
    Build QualityAssessment.build
    Configuration QualityAssessment.conf
}
```

Figure 4. DSLs to define and represent data and services

The viewpoint DSL is however more complex than the architecture framework DSL. As show in Figure 4, the DSL is decomposed by aspects: i) Data for the definition of metamodel, ii) User interface for the data representation by user interfaces, iii) Diagram for the graphical representation of data, iii) Services for the definition of business rules, services, and parameters, iv) Build to automatically generate continuous integration scripts, and v) Configuration to tune the generation parameters of architecture framework and viewpoint artefacts. The abstract syntax is extensible. Thus, other aspects could be supported, such as the definition of constraints. At the concrete syntax level, a main textual grammar of viewpoint aggregates textual grammars of viewpoint aspects as depicted in the following picture.

```
Viewpoint QualityAssessment {
        name: "QualityAssessment"
        Data QualityAssessment.data
        UI QualityAssessment.ui
        Diagrams QualityAssessment.diagram
        Services QualityAssessment.services
        Build QualityAssessment.build
        Configuration QualityAssessment.conf
}
```

Figure 5. Example of main editor of a viewpoint

The following figure exemplifies another editor for the definition of viewpoint metamodel. All the common features to describe a metamodel are covered. The words “ecore” and “capella” identifies external metamodels, respectively the Eclipse EMF and Capella metamodels. The “extends” section means that the current metaclass extends the definition of identified metaclasses (e.g., a Logical Component is enriched by a QualityAssessment metaclass).

```
QualityAssessment.data {
  Class QualityAssessment {
    description: "Quality Assessment"
    icon: "QualityAssessment.gif"
    extends fa.AbstractFunction, la.LogicalComponent,
        pa.PhysicalComponent
    Attributes:
      maturityLevel type ecore.EString
      confidenceLevel enum ConfidenceLevel
      assessed type ecore.EBoolean
    Associations:
      basedOn refers [0,*] QualityAssessment
      context refers [0,*] external capella.NamedElement
      measures contains [0,*] QualityMeasure
  }
  Class QualityMeasure {
    icon: "QualityMeasure.gif"
    Attributes:
      criterion type ecore.EString
      measureValue type ecore.EInt
  }
  Enumeration ConfidenceLevel {
    "Not Assessed" , Low , Medium , High
  }
}
```

Figure 6. Example of Data editor of a viewpoint

A generation function produces artefacts from the complete set of descriptions which conform to the abstract syntax and stored in the form of Eclipse EMF models. This mapping is realized by software factories [5] with EGF [3] to mass produce artefacts, such as code, but also models, diagrams or Eclipse plugins. The software factories are selected according to a target application which declares all the parameters and resources to target a specific platform (e.g., a targeted DSL, or UML; a tooling platform; collaborative work or not).

4. Lessons learned

Kitalpha incubated at Thales for several years before being recently open sourced in the framework of the PolarSys working group. At this stage, it is the appropriate time to present lessons learned about DSL in this development context.

Productivity and quality The combination of DSL and generation has dramatically improved productivity of the developers to implement viewpoints. Days become hours of development. For instance, for a development of a metamodels, user interface, diagrams, structure services, and continuous integration scripts, before it took about 8-10 days, and now about 5-8 hours. Worse before, there was not a systematic practice of continuous integration, and sometimes it remained manual. With the feedback from user teams, a central team has set up all the foundations, automated code production, solved code issues, and defined architectural rules of the produced artefacts. Boring activities, such as writing code of user interfaces, scripts for continuous integration, or definition of the Eclipse plugin dependencies are achieved with very few tuning. At this stage, the designers/developers are very satisfied by a textual syntax for its efficiency (e.g., with highlighted text, assistance, validation rules, but also for precise and accurate descriptions) and they are not on demand of another concrete syntax (e.g., graphical). About maintenance, the generators were designed to support incrementally. For instance, for the user interface description by DSL, there is a Java code merger during the translation phase in order to preserve the manual code; some artefacts, such as a build model to produce the continuous integration scripts, are replaced. The issue of migration, when viewpoint metadata evolve, has not been solved. Migration code is manually maintained, which is a lack. Kitalpha is based on Eclipse and tries to use the best tools for each aspect of development. For instance, for diagrams, Sirius dramatically reduces the complexity of GMF and there is a direct translation from the diagram DSL to a Sirius model. For the user interface aspect, the existing solution is based on a home-solution; the next step will be the adoption of PMF [12] which is a more powerful solution, with a real and rich metamodel, and with the ability, for evolution, to target multiple platforms (e.g., XWT, Web).

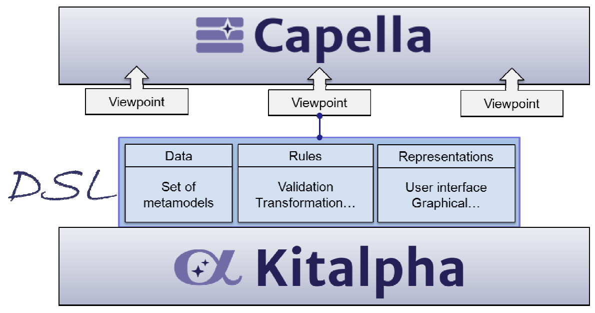


Figure 7. Enrichment of Capella with viewpoints

Homogeneity with DSL-based workbenches The primary need of Kitalpha was to develop viewpoints for Capella, previously named Melody Advance in its non-open source version, in order to extend Capella with new kinds of data, representations and rules, as depicted in Figure 7. Capella is a complex MBE workbench to describe architecture in system engineering. Capella is based on the DSL technique in order to accurately address and represent this domain for demanding system engineers. The first interest of the homogeneity between Kitalpha and Capella is to share common development foundations, what enables economies of scales. The second interest is that the enrichment of Capella with viewpoints developed with Kitalpha is seamless.

Achievement of the concrete syntax The combination of DSL and generation encapsulates complexity that users generally ignore. Concrete syntax is the top of the iceberg and must be as perfect as possible. The syntax, textual here, must be clear, light, with clear messages, completion, especially with predefined pieces of code adapted to the context of work. Functionally, the syntax to describe an aspect, for instance metamodels or diagrams, must be complete else there is a risk of complete rejection because it will be judged as a general lack of the tool. Finally, it appears that obtaining a mature textual syntax is time-consuming, what must not be underestimated.

Dynamic grammar extensibility One requirement was to have a dynamic extensibility of the textual syntax. At the abstract syntax level, it is very well managed. At the concrete syntax level, if it is possible to contextually adapt the textual syntax built with Xtext [16], the flexibility to extend it is not easy because the syntax and editors are compiled.

Separation of description by aspects Historically, the separation by aspects was born for a scalability reason: a complete viewpoint could not be described in one editor, otherwise it would have been too long with heterogeneous information. This separation enables to separate the different concerns and to have a modular organization of the abstract and concrete syntax. This separation by aspects is made possible by the extensibility of grammar. A main grammar is enriched by contribution of grammars (e.g., availability of new aspects, enrichment of the abstract or concrete syntax with new assistants and validation rules).

Impacts of decoupling by architecture framework and viewpoints The conformance to the ISO/IEC 42010 standard is structuring for the development practices, i) at the tooling level with dedicated DSL editors, generators, and packaging configurations, ii) for the architects, designers and developers when they describe a system. Finally, it appears that a traditional modelling development would correspond to one architecture framework with a big viewpoint. Separation of concerns becomes a best practice to decouple the development activities by viewpoints.

Question about a textual syntax for Sirius Sirius [14] is an Eclipse component for model representations, especially graphical with diagrams. The question is why there is an alternative to the Sirius tree editor to design diagrams. Firstly, the textual description of diagrams is needed for the integration with the other aspects described in a textual form. Secondly, the Kitalpha editor simplifies some parts in comparison with Sirius and offers accelerators during textual completion. Thirdly, a textual notation enables to have a complete view of a diagram definition at a glance.

Tool-independence of the syntax The syntax to describe diagram is independent of Sirius but it is close. Actually, it is uneasy to be tool-independent and map directly inhomogeneous metamodels of a same aspect, for instance diagrams. It is the same for the other aspects, for instance user interfaces or continuous integration. However, regarding the data aspect, it is encouraging that either DSL or UML could be mapped. Moreover, for continuous integration, scripts are not directly generated from the DSL description but from an intermediary model which could evolve over the time in order to target other continuous integration environments.

Separation of the development tasks of abstract and concrete syntax In terms of team organization, the development tasks of the abstract and concrete syntax have been assigned to two different persons in order to take equally care of each of them. Validation has been assigned to a third person in order to be impartial.

5. Conclusion

Kitalpha is an environment to develop and execute MBE workbenches for system architecture description. This paper has presented the real advantage to use the DSL technique to develop accurately, quickly, and safely such MBE workbenches, and to realize economies of scales. The DSL technique has brought autonomy and efficiency to designers and developers in order to develop and maintain their own viewpoints. Kitalpha fills the gap in Eclipse with an integrated and pure DSL environment, and avoid using a multitude of tools, with glue between, which is a risk for new projects or without experimented practices. In the development context of co-engineering, Kitalpha is an enabler for architects and speciality engineers to seamlessly extend an architecture framework for a complete description of system architecture in system, software and hardware engineering.

Acknowledgments

We thank Stéphane Bonnet, Matthieu Helleboid, Thomas Guiu, and Faycal Abka for their collaboration. This initiative is supported by Sys2Soft, a French research project, and Crystal, a European project.

References

[1] Departement of Defense, *The DoDAF Architecture Framework Version 2.02*, August 2010

[2] Eclipse Working Groups, http://www.eclipse.org/org/workinggroups/

[3] EGF, Eclipse Generation Factories, http://eclipse.org/egf

[4] Exertier, D., Bonnet, S., *Arcadia / Capella, a field-proven modeling solution for system and software architecture engineering*, eclipsecon France 2014

[5] Greenfield, J., Short, K., Cook, S., and Kent, S., *Software Factories, Assembling applications with Patterns, Models, Framework, and Tools*, Wiley, 2004

[6] ISO/IEC/IEEE 42010, *Systems and software engineering — Architecture description*, First edition, 2011-09-15

[7] Kelly, S., Tolvanen, J.-P., *Domain-Specific Modeling*, IEEE Computer Society, Wiley-Interscience Publication, 2008

[8] Kitalpha, https://www.polarsys.org/projects/polarsys.kitalpha

[9] NATO Architecture Framework v4.0 Documentation, http://nafdocs.org/

[10] OMG Unified Modeling Language ™ (OMG UML), Version 2.5 FTF – Beta 1, ptc/2012-10-24

[11] Papyrus, https://www.eclipse.org/papyrus/

[12] PMF, Presentation Modeling Framework, https://wiki.eclipse.org/Pmf

[13] PolarSys, https://www.polarsys.org

[14] Sirius, http://eclipse.org/sirius/

[15] Voelter, M., *DSL Engineering*, dslbook.org, 2013

[16] Xtext, http://www.eclipse.org/Xtext/