

PROPRIETARY RIGHTS STATEMENT

THIS DOCUMENT CONTAINS INFORMATION, WHICH IS PROPRIETARY TO THE CRYSTAL CONSORTIUM. NEITHER THIS DOCUMENT NOR THE INFORMATION CONTAINED HEREIN SHALL BE USED, DUPLICATED OR COMMUNICATED BY ANY MEANS TO ANY THIRD PARTY, IN WHOLE OR IN PARTS, EXCEPT WITH THE PRIOR WRITTEN CONSENT OF THE CESAR CONSORTIUM THIS RESTRICTION LEGEND SHALL NOT BE ALTERED OR OBLITERATED ON OR FROM THIS DOCUMENT. THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N° 332830 AND FROM SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.



CRritical **SY**STem Engineering **Acce**Leration

WP 6.9 - Specification, Development and Assessment Report

D609.001

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	WP 6.9 - Specification, Development and Assessment Report
Deliverable No.	D609.001
Dissemination Level	CO
Nature	R
Document Version	V1.0
Date	2014-04-30
Contact	Stéphane LACRAMPE
Organization	Obeo
Phone	+33 2 51 13 51 42
E-Mail	Stephane.lacrampe@obeo.fr

AUTHORS TABLE

Name	Company	E-Mail
Stéphane LACRAMPE	Obeo	Stephane.lacrampe@obeo.fr
Jerome Lenoir	Thales R&T	jerome.lenoir@thalesgroup.com
Daniel Exertier	Thales Global Services	daniel.exertier@thalesgroup.com
Benoît Langlois	Thales Global Services	benoit.langlois@thalesgroup.com
Olivier Constant	Thales Global Services	olivier.constant@thalesgroup.com
Yves Yang	Soyatec	yves.yang@soyatec.com

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
1.0	01/04/2014	Version submitted for review	

CONTENT

D609.001	I
1 INTRODUCTION.....	7
1.1 ROLE OF DELIVERABLE	7
1.2 RELATIONSHIP TO OTHER CRYSTAL DOCUMENTS	7
1.3 STRUCTURE OF THIS DOCUMENT	7
2 OPEN SOURCE INTEGRATED ENVIRONMENT FOR THE DEVELOPMENT OF MODEL-BASED ENGINEERING (MBE) SOLUTIONS BRICK.....	9
2.1 DESCRIPTION	9
2.1.1 <i>Manual</i>	11
2.2 USE CASE COVERAGE AND APPLICATION	11
2.3 GENERAL IMPROVEMENT	11
2.4 INTEGRATION AND INTEROPERABILITY	14
3 OPEN SOURCE COMPONENT FOR EDITING MODELS BY GRAPHICAL VIEWS AND WEB RENDERING BRICK.....	16
3.1 DESCRIPTION	16
3.1.1 <i>Manual</i>	18
3.2 USE CASE COVERAGE AND APPLICATION	18
3.3 GENERAL IMPROVEMENT	18
3.4 INTEGRATION AND INTEROPERABILITY	25
4 OPEN SOURCE COMPONENT FOR GENERATING GUI PRESENTATION OF BUSINESS DATA BRICK	28
4.1 DESCRIPTION	28
4.1.1 <i>Manual</i>	29
4.2 USE CASE COVERAGE AND APPLICATION	30
4.3 GENERAL IMPROVEMENT	30
4.4 INTEGRATION AND INTEROPERABILITY	33
5 OPEN-SOURCE COMPONENTS FOR MODEL TRANSFORMATION, MODEL-DRIVEN CODE GENERATION AND MODEL VALIDATION AND ANALYSIS BRICK.....	34
5.1 DESCRIPTION	34
5.1.1 <i>Manual</i>	37
5.2 USE CASE COVERAGE AND APPLICATION	37
5.3 GENERAL IMPROVEMENT	37
5.4 INTEGRATION AND INTEROPERABILITY	45
6 MODEL CO-EVOLUTION BRICK.....	46
6.1 DESCRIPTION	46
6.1.1 <i>Manual</i>	47
6.2 USE CASE COVERAGE AND APPLICATION	47
6.3 GENERAL IMPROVEMENT	47
6.4 INTEGRATION AND INTEROPERABILITY	53
7 INTEGRATION OF REQUIREMENT MANAGEMENT FOR MULTI-VIEWPOINT ENGINEERING BRICK	55
7.1 DESCRIPTION	55
7.1.1 <i>Manual</i>	55
7.2 USE CASE COVERAGE AND APPLICATION	55
7.3 GENERAL IMPROVEMENT	55

7.4	INTEGRATION AND INTEROPERABILITY	57
8	TERMS, ABBREVIATIONS AND DEFINITIONS	58
9	REFERENCES.....	59
10	ANNEX	61
10.1	ANNEX I: STATE OF THE ART OF MODEL SYNCHRONIZATION	61
10.1.1	<i>Organizational aspects.....</i>	<i>65</i>
10.1.2	<i>Establishing semantical correspondences between models.....</i>	<i>72</i>
10.1.3	<i>Concurrency and consistency control.....</i>	<i>98</i>
10.2	ANNEX II: CO-EVOLUTION SPECIFICATION	112
10.2.1	<i>Problem</i>	<i>112</i>
10.2.2	<i>Positioning Overview.....</i>	<i>112</i>
10.2.3	<i>Requirements.....</i>	<i>113</i>

Content of Figure

Figure 10-1: Model transformation for code generation	61
Figure 10-2: Data Warehouse	62
Figure 10-3: Architecture metamodel - ISO 42010.....	66
Figure 10-4: process-based approach.....	67
Figure 10-5: Single-metamodel approach	68
Figure 10-6: Multiple metamodels approach	69
Figure 10-7: Weaving the meta-models	73
Figure 10-8: AMW Core Metamodel.....	74
Figure 10-9: OpenFlexo intermediate metamodel.....	75
Figure 10-10: Merging at metamodel level	77
Figure 10-11: Example of VIATRA Rete network.....	78
Figure 10-12: Transformation rules usage	80
Figure 10-13: Model-to-model transformation features.....	81
Figure 10-14: ATL transformation rule.....	85
Figure 10-15: A relation in QVTR	87
Figure 10-16: VCTL transformation (create a reverse connection).....	90
Figure 10-17: EOL matching query	91
Figure 10-18: ETL M2M transformation.....	92
Figure 10-19: TransML transformation framework	93
Figure 10-20: TransML framework traceability links.....	96
Figure 10-21: Centralized, heavy client, proxy-based architecture	99
Figure 10-22: Pair-to-pair, heavy client architecture	100
Figure 10-23: Multi-cluster architecture	102
Figure 10-24: Three-way merge	104
Figure 10-25 : Differential Synchronization workflow	105
Figure 10-26: Physical network model	106
Figure 10-27: High-level services model	107
Figure 10-28: Modified physical network model	107
Figure 10-29: Harmonized high-level services model	107
Figure 10-30: Development compatibility inconsistencies.....	109

Content of Table

Table 8-1: Terms, Abbreviations and Definitions	58
Table 2-2: classification of solutions.....	71

Version	Nature	Date	Page
V01.00	R	2014-04-2830	6 of 117

1 Introduction

1.1 Role of deliverable

In this deliverable, Bricks are documented which are developed in the WP6.9. This deliverable is updated iteratively, i.e. three times during the project runtime, based on the corresponding milestones of the project. Therefore the Brick documentations in this document conform to an evolutionary process of continuous development and enhancement of the CRYSTAL solutions and are complementary to previous versions of this deliverable.

1.2 Relationship to other CRYSTAL Documents

This work package is connected to:

- The aerospace use case WP207 which will experiment bricks produced in WP609
- The IOS provider : WP601 : IOS Evolution & Development, Standardisation
- The coordination of R&T activities : WP600

1.3 Structure of this document

This document is broken down into 6 main parts corresponding to the 6 bricks that are developed under WP6.9:

- MBE Development & Execution Environments
- Open Source component for editing models by graphical views and Web rendering
- Open Source component for generating GUI presentation of business data
- MBE technologies
- Model co-evolution
- Integration of requirement management for multi-viewpoint engineering

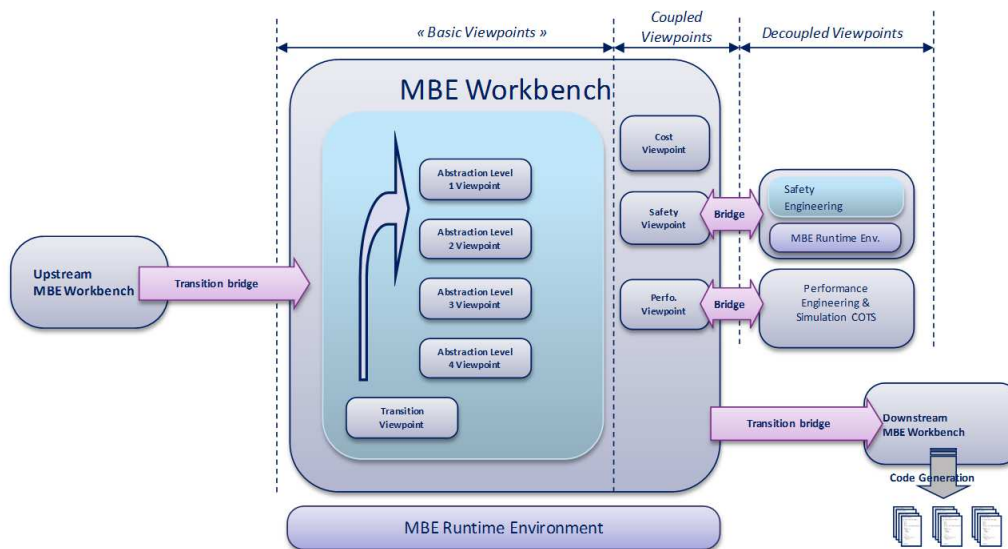
Annexes are used to provide some actual content of documentary work achieved in the project.

Introduction to WP609 context and objectives

As Systems and software complexity is increasing, it requires appropriate means to describe and design these systems. To define the architecture of a system, the various stakeholders with their own concerns, contribute to its description. For instance, the safety engineer does not have the same concerns as the head of product line. An architecture description allows everyone to understand and demonstrate that the architecture of the system meets its concerns, and their related requirements. The major reference for specifying how the architecture descriptions are expressed is the standard [ISO / IEC 42010] 1 published in July 2011.

The objective of this work package is to provide a set of bricks (the Core Technology Kit) for building and executing Integrated Engineering Model Based Environments (IEMBE). An IEMBE is composed of a set of tools allowing engineers to define the architecture, design, develop a system and / or its components (subsystems, software, equipment, etc..) for a given domain, as well as dedicated viewpoints to deal with aspects such as dependability, performance, etc.

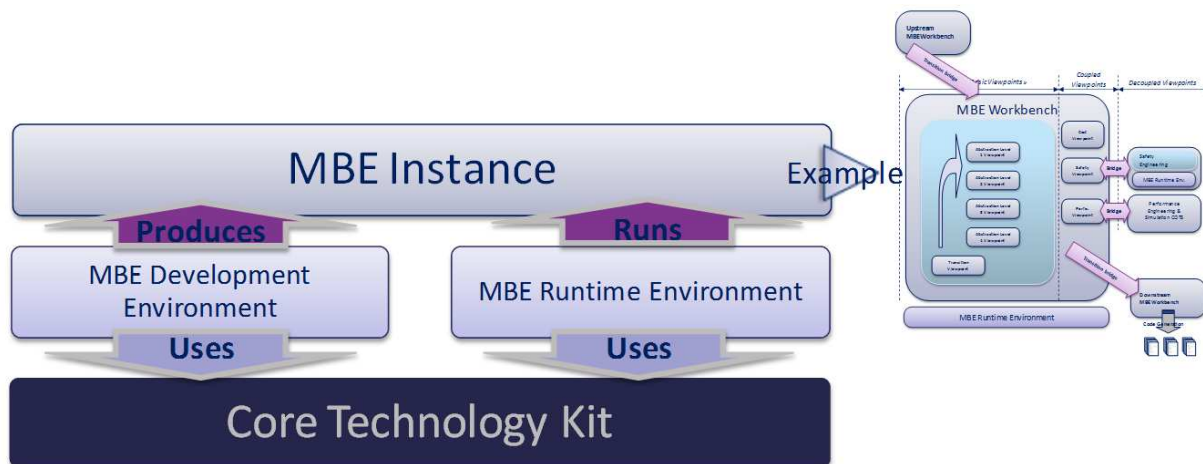
Version	Nature	Date	Page
V01.00	R	2014-04-2830	7 of 117



Example of Integrated Engineering Model Based Environment

The Core Technology Kit supports two levels of usage as shown in the figure above:

- the MBE Development Environment when the IEMBE is configured
- the MBE Runtime Environment when the IEMBE is used



General architecture schema

The Core Technology Kit supports development and implementation of major types of tools for model driven engineering : graphical editing of models, data entry model, model transformation, generation of information from modelling, verification and analysis of models, synchronization between models.

These components must be integrated, consistent, and provide capabilities to be used in a multi-viewpoint environment (extensibility, inheritance, composition, etc.) or used independently.

While the Core Technology Kit and its components have a meaning as Eclipse projects, some components or the IEMBE itself may fit better within the Polarsys platform (<http://wiki.eclipse.org/Polarsys>), whose guidelines are set by industry in coordination with tool providers and academics.

2 Open source integrated environment for the development of model-based engineering (MBE) solutions brick

2.1 Description

Name:	Benoît Langlois
Contact:	benoit.langlois@thalesgroup.com - +33 1 70 28 23 53
Technical Information:	This brick is an industrial maturity integrated environment dedicated to the development and execution of architecture frameworks and viewpoints. This environment implements the ISO/IEC 42010 standard [ISO/IEC FCD 42010, 2010].
Dependencies	This brick is based on the Eclipse platform and depends on Eclipse components such as EMF, Sirius, and Xtext.
License	This brick is provided as an open source component under the Eclipse Platform License (EPL V1.0 - http://www.eclipse.org/legal/epl-v10.html).
Additional information	Brick website: An Eclipse component was created for this brick in April 2014. The website address will be http://www.eclipse.org/kitalpha/ The reference of Kitalpha in PolarSys: https://www.polarsys.org/projects/polarsys.kitalpha

This MBE workbench development and execution environment is named Kitalpha.



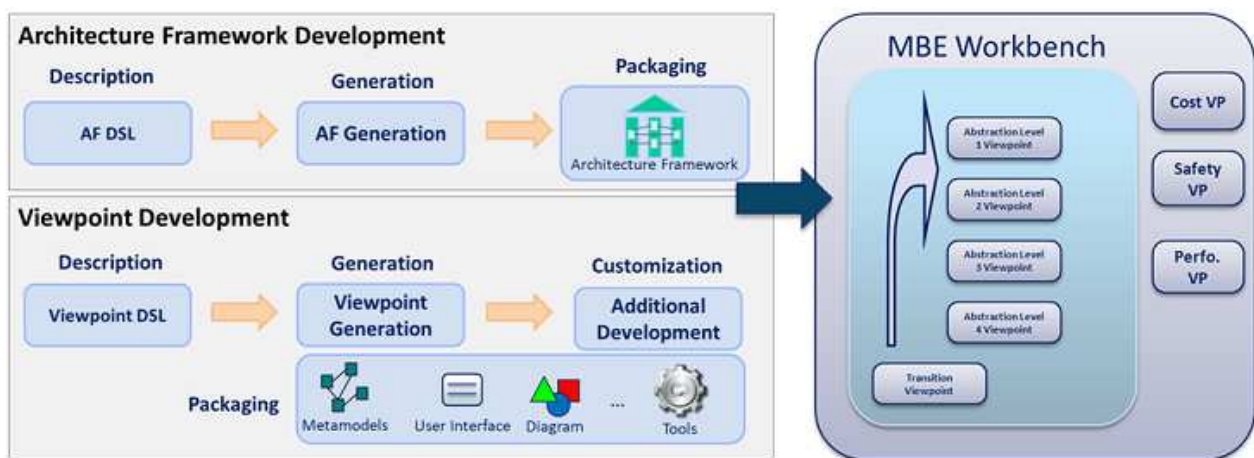
Kitalpha is a modelling environment which implements the ISO/IEC 42010 standard for system description in system and software engineering. It provides both a development and runtime environment to create and execute rich MBE (model-based engineering) workbenches (e.g., edition with diagrams, documentation, import/export, model transformation / analysis / validation) for system / software architects and engineers in small- to large-scale projects.

For reusability, the development environment contains a kit of MBE core components, *i.e.* a set of engineering components common to different variants of MBE workbenches such as a semantic browser, model transformation, model-to-text generation, model analysis & verification, or import/export functions.

Conforming to this standard, an MBE workbench is an architecture framework which aggregates viewpoints clearly separating concerns (e.g., performance, safety, security, cost). A viewpoint is an engineering extension which comes with its own 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 defines the complete description of a system.

Version	Nature	Date	Page
V01.00	R	2014-04-2830	9 of 117

To build MBE workbenches, designers must also be able and autonomous to create and maintain their own viewpoints dedicated to their own needs and concerns. The Kitalpha main objective is to ease the development of viewpoints, without coding, that developers can enrich afterward, for instance for algorithm implementation. To meet this requirement, Kitalpha offers a development environment made of DSLs (Domain-Specific Languages) to assist designers and developers in their architecture frameworks and viewpoints development activities. For instance, textual editors make it possible to declare viewpoint metamodels, user interfaces, diagrams, or services. From those DSLs, generators build all the architecture framework and viewpoint artefacts. For example, the declaration of diagrams using DSLs becomes the technical description of Sirius diagrams. During the stages of edition with DSLs and generation, the notion of target application is introduced to manage the variability of environments in which the artefacts are to be deployed and executed (e.g., DSL vs. UML, CDO vs. XMI environments).



Development Process Overview

The Kitalpha development environment also provides MBE core components to develop complete MBE workbenches, for instance for the functions of import/export (e.g., for data exchange with MBE workbenches or specialty tools), model transformations with traceability, or html documentation generation, plus the other MBE Core components that are provided as other technical bricks in this WP (i.e. Sirius presented in Section 2; PMF in Section 4; Composer, Transposer, Accuracy in Section 5; Model Co-Evolution in Section 6).

Kitalpha potentially has the ability to implement architecture framework standards (e.g., TOGAF/MODAF) but also proprietary method or domain workbenches.

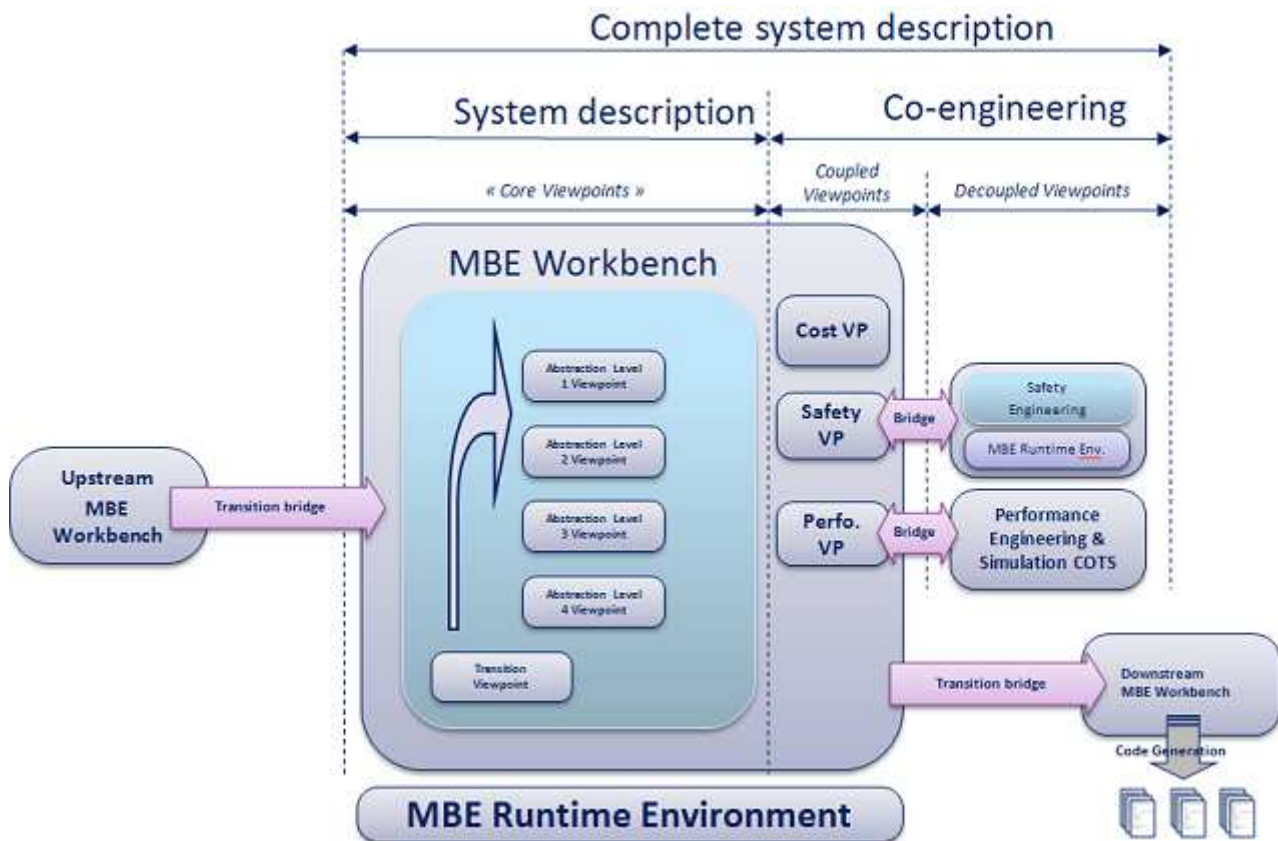


Figure 3. Illustration of MBE workbench

Link to the Kitalpha proposal: <http://eclipse.org/proposals/polarsys.kitalpha/>

2.1.1 Manual

This part will be written when the Eclipse component will be created and its web site available.

2.2 Use Case coverage and application

This brick will be used in the context of the use case 2.7 (Thales Alenia Space future satellite platform) with a focus on producing a dedicated multi-viewpoint engineering environment, using the definition of this new avionics platform as a use case.

Further usage of this brick by other Crystal Use Cases will have to be identified.

2.3 General Improvement

The specification of this brick was written in the context of the French Sys2Soft project. It can be found on the AVL Sharepoint for the Crystal project

https://projects.avl.com/11/0154/Data%20Exchange/007_Work/SP6_RnT_Activities/6.9%20Multi-viewpoint%20Engineering/Sys2Soft-WP2.1-MBE%20Environment-1.1.0.docx

Version	Nature	Date	Page
V01.00	R	2014-04-2830	11 of 117

TI NAME: Multi-viewpoint Environment - State Of the Art					
TI_ID	TI_0048	Kind of TI	G	Contact email	Jerome.lenoir@thalesgroup.com
Description: <p>The purpose of this TI is to provide a State of the Art about techniques and frameworks allowing language designers to define Multi-viewpoint Environment. The use of multiple views has become standard practice in industry. A survey on the industrial needs from architectural languages revealed that 85% of the 48 interviewed practitioners use multiple views when architecting a software system. Current frameworks are defined with varying degrees of rigor and offering varying levels of automated tool support. However, architecture frameworks tend to be closed; that is, architecture frameworks mainly focus on a closed set of stakeholder concerns, viewpoints, ADLs, etc.</p> <p>This state of the art will be provided in the next release of this document.</p>					
Link to internal working documents:					

TI NAME: Prototype of the MBE development environment (MBE DE)					
TI_ID	TI_0095	Kind of TI	T	Contact email	benoit.langlois@thalesgroup.com
Description: <p>The work achieved in April 2014:</p> <ul style="list-style-type: none"> The MBE DE provides textual DSLs to describe architecture frameworks and viewpoints. The supported aspects which enable to describe a viewpoint are: 1) Data for description of metamodel, 2) User Interface for description of user interfaces, 3) Diagram for description of diagrams, 3) Services for definition of business rules and services which orchestrates business rules, 4) Build for code building of viewpoint, 5) Configuration for parameterization of the generation from a textual description. The MBE DE generates executable architecture framework and viewpoint artifacts from textual descriptions. For deployment, the MBE DE automatically packages architecture framework and viewpoint artifacts. <p>The Innovations are:</p> <ul style="list-style-type: none"> Implementation of the ISO/IEC 42010 standard [ISO/IEC FCD 42010, 2010]. The DSL technique has several advantages: 1) The architecture framework and viewpoint DSLs enable to easily and quickly create architecture frameworks and viewpoints, 2) Development of viewpoint is no longer restricted to developers but becomes accessible to designers, 3) Generation defines a standardized backbone (e.g., naming and contents of Eclipse plugins, organization of code, models, generators) which drives developers in their development activities. An architecture framework and its viewpoints have their own policies and use specific components (e.g., metamodels are either based on EMF or UML, user interfaces are either based on SWT, EEF or XWT). The notion of Target Application defines a set of common properties for parameterization of the DSLs and generators. This enables to introduce variability in the definition and generation of viewpoints. For instance, from the same declaration of viewpoint data, it becomes possible to either generate a model API in the EMF context or a UML profile in the context of UML. 					

The issues are:

- The definition of a textual grammar must be adopted by designers and developers.
- The architecture of the MBE DE must meet the needs of 1) Evolutivity in order to address new and evolving MBE environments, 2) Sustainability, 3) Flexibility while the textual DSLs and generators are complex and sophisticated.
- The brick for user interfaces described in Section 4 is really expected to support advanced user interfaces.

The results are:

- The availability of DSLs for the description of architecture frameworks and viewpoints.
- The availability of generators for architecture frameworks and viewpoints which target the Eclipse EMF environment.
- The compatibility of Sirius for the DSL of diagram description and generators which produce Sirius odesign files.

Link to internal working documents:

TI NAME: Prototype of the MBE execution environment (MBE EE)					
TI_ID	TI_0096	Kind of TI	T	Contact email	benoit.langlois@thalesgroup.com
Description: <p>The work achieved in April 2014:</p> <ul style="list-style-type: none"> • The MBE EE enables to execute the artifacts produced by the MBE DE. • The MBE EE offers the capability to dynamically extend EMF-based viewpoints. • The MBE EE supports the following services: 1) activation of viewpoints with the related viewpoint elements (e.g., viewpoint data, user interfaces, and diagrams), 2) deactivation of the same viewpoint elements. <p>The Innovations are:</p> <ul style="list-style-type: none"> • The extensibility of application by viewpoints. A viewpoint can be considered as an engineering block with all the aspects it contains (e.g., metamodel, user interfaces, diagrams, services). • The support of activation / deactivation services of viewpoints. • EMF (Eclipse Modeling Framework) only supports metamodel extension by the mechanism of inheritance between metaclasses. A mechanism of "extension by aggregation" was introduced, with a specific metamodel named eMDE, in order to extend a metaclass with a set of metaclasses. Then, in the viewpoint context, a same metaclass can be viewed with several facets, for instance with Performance, Safety or Security information. <p>The issues are:</p> <ul style="list-style-type: none"> • The services of activation/deactivation require a mechanism of synchronization with the different components involved by a viewpoint (i.e., when the viewpoint is activated / deactivated, all the components must be synchronously activated / deactivated). Then, those components must have the ability to be activated / deactivated. For instance, this function is supported by Sirius with the mechanism of diagram layer. A Kitalpha viewpoint manager ensures this synchronization, for instance by the activation or 					

deactivation of Sirius diagram layers. However, not all the components have this ability to be activated or not.

The results are:

- The services of Installation, activation, and deactivation of viewpoint are supported.
- In the EMF context, the supported viewpoint aspects are: metamodel, diagram, services, build, and configuration. A Thales prototype validated the services of activation and deactivation for the aspect of user interface.
- The availability of a simple use case named "Simple Component", for description of Software and Hardware components. The purpose of this use case is to validate the creation, generation, deployment, and use of viewpoints. The "Simple Component" viewpoint is extended by "Safety" and "Performance" viewpoints. The viewpoint metamodels are based on EMF; a Sirius diagram presents a "Simple Component" model and is enriched by Safety and Performance layers. The activation / deactivation are operational in this context.

Link to internal working documents:

2.4 Integration and Interoperability

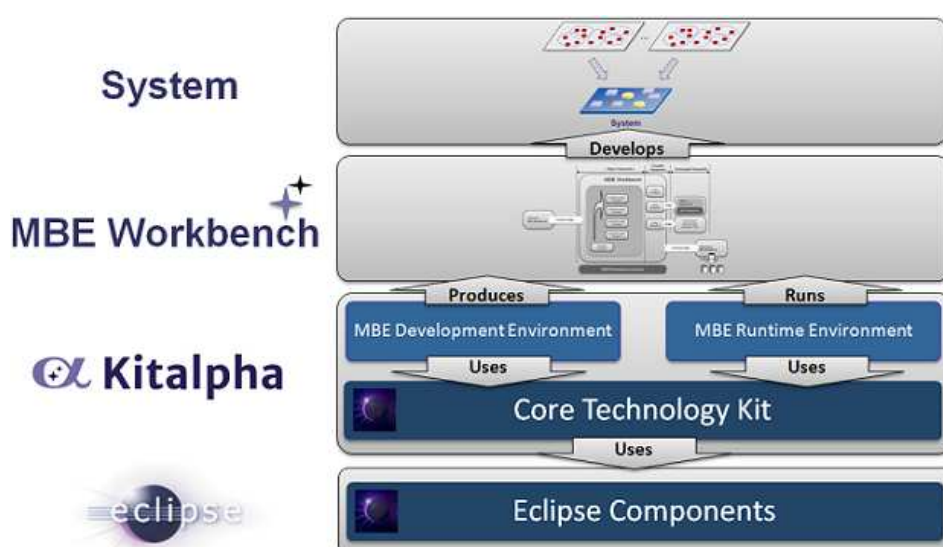
TI NAME: Core Technology Kit

TI_ID	TI_0075	Kind of TI	T	Contact email	benoit.langlois@thalesgroup.com
-------	---------	------------	---	---------------	---------------------------------

Description:

Kitalpha enables to develop and execute MBE workbenches, such as a tool dedicated to system engineering. The following picture presents the different working contexts with regard to Kitalpha.

1. At the top, stakeholders describe a system architecture,
2. An MBE workbench enables designers to describe system architecture,
3. Kitalpha is a workbench that allows developing and executing MBE workbenches,
4. Finally, Kitalpha is built atop Eclipse and mainly uses modeling components.



Internally, Kitalpha is divided into two layers:

- An Engineering Layer which is a set of components and services for developing and executing MBE workbenches. This layer contains all the architecture framework / viewpoint metamodels, DSLs with their representations, and services.
- A Core Technology Kit (CTK) which is a set of technical components and frameworks required by the Engineering Layer.

The CTK is dedicated to host a set of added-value technological components which are not provided by Eclipse components or to adapt existing tools for a specific purpose (e.g., adaptation of the standard EGF factories).

The work achieved in April 2014:

- The CTK integrates MBE Tools needed at the core of the MBE DE and EE but they are generic enough to be reused in other MBE contexts. At this stage, there exist: 1) A Reporter to report messages (e.g., Error, Warning, Information or new kinds of messages) organized by topics, for instance by viewpoint name, 2) A "Resource Reuse" tool to find a resource (e.g., model, document) by its name, version, either in the Eclipse workspace or platform.
- The CTK integrates MBE Tools needed by the MDE EE, for instance to support the functions of import / export of models, model transformation and generation, model validation. For this, the CTK integrates Composer, Transposer, Cadence and Accuracy, which are described in Section 5.
- The CTK integrates metamodels of reference needed at the core of MBE DE and EE but they are generic enough to be reused in other MBE contexts. At this stage, only one metamodel exists with its API: the eMDE metamodel to support the mechanism of "extension by aggregation".

The Innovations are:

- The elaboration of a common and unified set of MBE tools.
- While the Eclipse Modeling project only provides technical components, the purpose of the CTK is twofold: 1) providing components absent in the Modeling project, 2) providing components with engineering concerns, such as a semantic browser.
- Kitalpha can integrate a technology that produce connectors (conforming the IOS of the Crystal RTP) providers and consumers for interoperability of the produced MBE environment, although this brick is not identified in the project.

The issues are:

- Avoiding an overlap between CTK components.
- A selection of pertinent, valuable, and sustainable components.
- Coordination between contributors.

The results are:

- An initialization of the CTK with components used by the MBE DE and EE.
- A physical integration of Composer, Transposer and Accuracy components.
- Integration with external components, and especially Sirius today for the graphical and tabular representation of models.

Link to internal working documents:

3 Open Source component for editing models by graphical views and Web rendering brick

3.1 Description

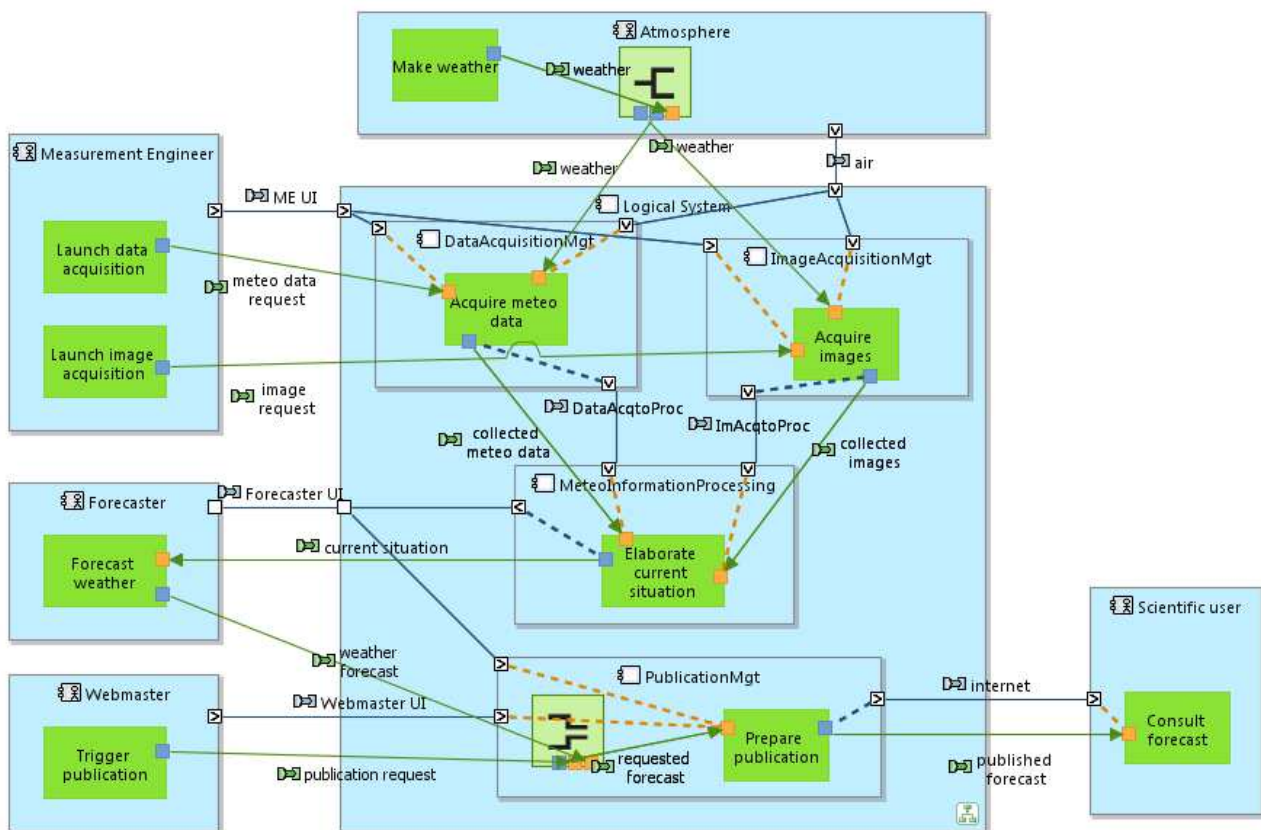
Name:	Stéphane LACRAMPE
Contact:	Stephane.lacrampe@obeo.fr - +33 2 51 13 51 42
Technical information:	This brick is an industrial maturity component for the development and execution of graphical model editors. The component allows defining and executing graphical or tabular modellers for instance, and supporting layers and filters management, collaborative modelling, etc.
Dependencies	This brick is based on the Eclipse platform and depends on Eclipse components like EMF and GMF.
License	This brick is provided as an open source component under the Eclipse Platform License (EPL V1.0 - http://www.eclipse.org/legal/epl-v10.html) except for the Web rendering part of the brick (license TBD).
Additional information	Brick website : http://www.eclipse.org/sirius/

This brick has been made available as an open source project called Sirius. More information can be found here: <http://www.eclipse.org/sirius/>



Sirius is an Eclipse project which enables the specification of a modelling workbench in terms of graphical, table or tree editors with validation rules and actions using declarative descriptions. All shape characteristics and behaviours can be easily configured with a minimum technical knowledge. This description is dynamically interpreted to materialize the workbench within the Eclipse IDE. No code generation is involved; the specifier of the workbench can have instant feedback while adapting the description. Once completed, the modelling workbench can be deployed as a standard Eclipse plugin. Thanks to this short feedback loop a workbench or its specialization can be created in a matter of hours.

Sirius has been created by Thales and Obeo to provide a generic graphical modelling workbench for model-based architecture engineering that can be easily tailored to fit specific project needs in complex environments.



Principles of Sirius

A modeling workbench created with Sirius is composed of a set of Eclipse editors (diagrams, tables and trees) which allow the users to create, edit and visualize EMF models. The editors are defined by a model which defines the complete structure of the modeling workbench, its behavior and all the edition and navigation tools. This description of a Sirius modeling workbench is dynamically interpreted by a runtime within the Eclipse IDE.

For supporting specific need for customization, Sirius is extensible in many ways, notably by providing new kinds of representations, new query languages and by being able to call Java code to interact with Eclipse or any other system.

An overview of the technology can be viewed here:

https://www.youtube.com/watch?feature=player_embedded&v=hyDxSmbSi2g and here :

https://www.youtube.com/watch?feature=player_embedded&v=SEfhNI47iA and a 4 minutes tutorial is available here :

https://www.youtube.com/watch?feature=player_embedded&v=08XIW8hZOyQ

The specification of this brick is made in the context of the Sys2Soft project and is contributed as background to the Crystal project. It can be found on the AVL Sharepoint for the Crystal project:

https://projects.avl.com/11/0154/Data%20Exchange/007_Work/SP6_RnT_Activities/6.9%20Multi-viewpoint%20Engineering/Sys2Soft-WP2.2-D2.2_1-SiriusSpecification.pdf

An updated version of this specification will be contributed in May 2014.

Version	Nature	Date	Page
V01.00	R	2014-04-2830	17 of 117

3.1.1 Manual

This brick can be installed directly from the Sirius Website by following this link: <http://www.eclipse.org/sirius/download.html>. There are 3 different and simple ways that are provided to install Sirius.

A tutorial can be found here: <http://www.eclipse.org/sirius/getstarted.html>.

An extensive documentation is available online (<http://www.eclipse.org/sirius/doc/>) as well as embedded within the brick. It explains how to use Sirius (user Manual) but also how to customize Sirius (Specifier manual).

An active forum is available: http://www.eclipse.org/forums/index.php?t=thread&frm_id=262 and bugs can be reported here: https://bugs.eclipse.org/bugs/buglist.cgi?classification=Modeling&list_id=6776579&product=Sirius&query_format=advanced.

Finally, source code is available here: <http://git.eclipse.org/c/sirius/org.eclipse.sirius.git>.

It is interesting to note that the brick has been presented several times jointly by Obeo and Thales within international conference like Eclipse Con Europe in October 2013 and Eclipse Con America (San Francisco) in March 2014.

3.2 Use Case coverage and application

This brick will be used in the context of use case 2.7 (Thales Alenia Space future satellite platform) with a focus on evaluating multi-viewpoint engineering capabilities, using the definition of this new avionics platform as a use case.

Main points that are to be assessed are:

- User friendliness of the tool and underlying concepts;
- Relevance of the viewpoints defined around the system model;
- Selection of the tooling for future extensions;
- Capabilities to link the system design level with the software design level.

Other use cases may use this brick but this still has to be identified.

3.3 General Improvement

TI NAME: Sirius - refactoring for initial contribution					
TI_ID	TI_0018	Kind of TI	T	Contact email	Stephane.lacrampe@obeo.fr
Description: This task has been the first done in the context of Sirius and preliminary to the other works. It is a global refactoring of the component code for open sourcing it. This includes an homogenization and compliance with the requirements of Eclipse projects and redistribution of certain features to meet the future business model.					

Because the code base is quite large and the structure of the plug-ins is quite complex, we have created an OSGi architecture refactoring tool to automate part of this task.

The Eclipse project has also been formally created and provisioned.

Another important part of the work consisted in creating and configuring the Eclipse infrastructure so that the Sirius component is compiled, tested and built everyday and made available to the community.

The first release of Sirius is v0.9, was delivered in December 2013.

Link to internal working documents:

TI NAME: Sirius – Multi Viewpoints Graphical Modelling environment - Prototype

TI_ID	TI_0019	Kind of TI	T	Contact email	Stephane.lacrampe@obeo.fr
-------	---------	------------	---	---------------	---------------------------

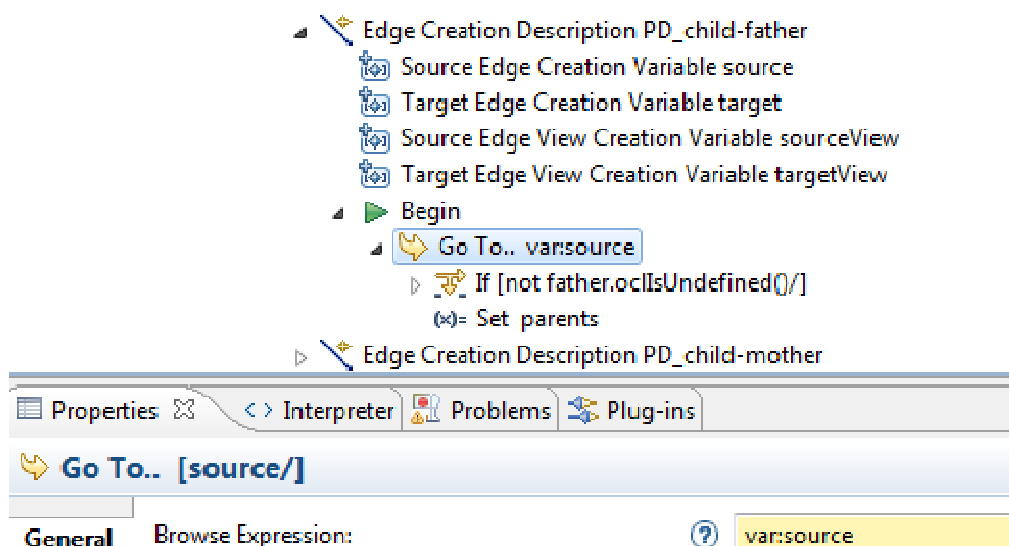
Description :

New expression interpreters to boost performances:

Expressions are used by Sirius to retrieve information from the model (to populate a representation, to dynamically compute graphical properties or to define the behavior of a tool). Before Sirius 0.9, expressions had to be written in Acceleo or standard OCL, even for retrieving very simple information.

Three new interpreters have been added to simply retrieve the value of features (attributes and relations), to evaluate services and to get variables.

To use these interpreters, just start the expression with the tags `feature:`, `service:` or `var:` followed by the name of the feature, service or variable.

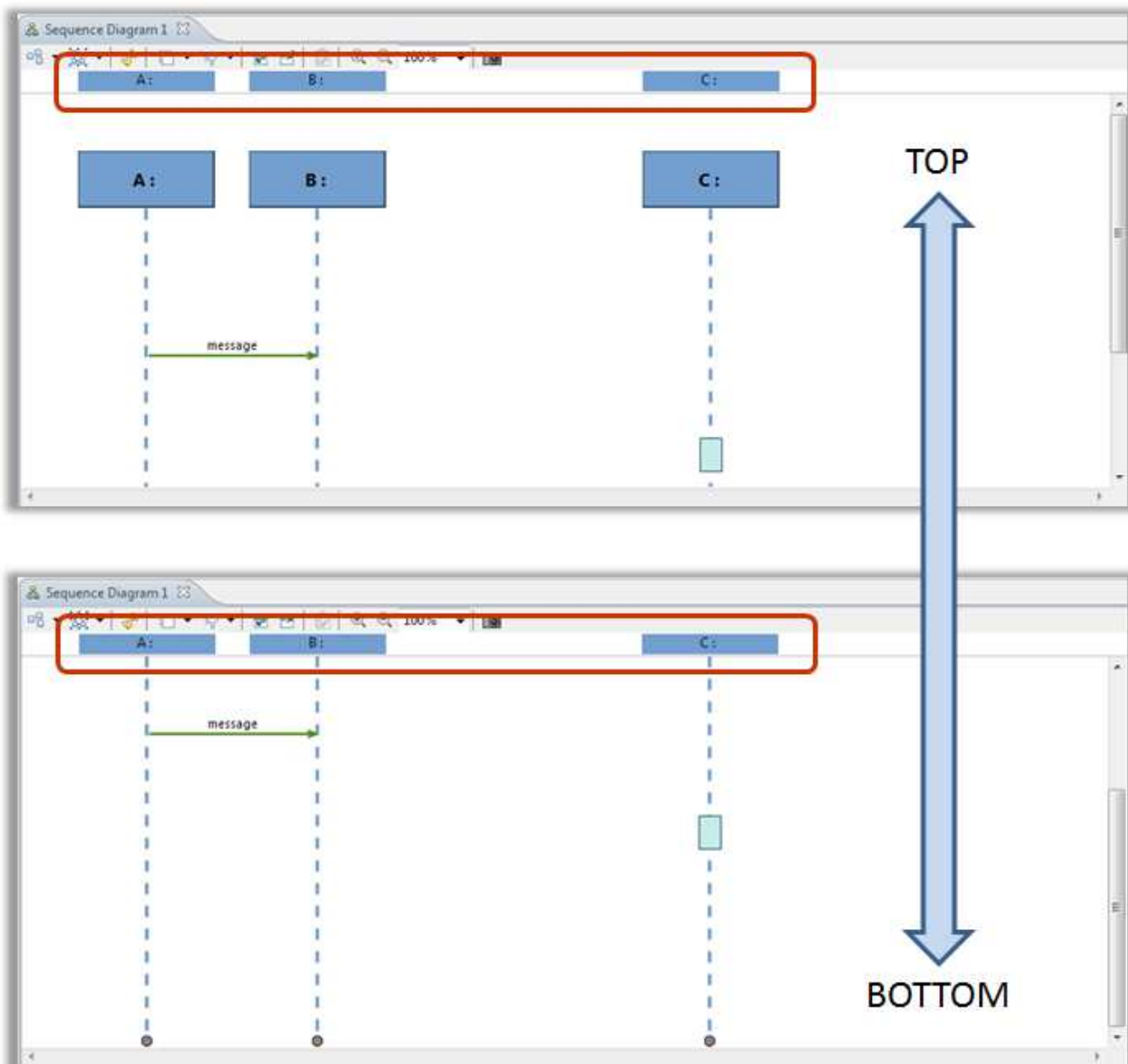


By using these interpreters, you can improve the performance of your modeler up to 30 %, as Sirius can directly evaluate simple expressions without having to delegate this work to Acceleo or OCL engines (which implies more controls).

Fixed diagram header:

On a sequence diagram, you can now choose to display a fixed lifeline header.

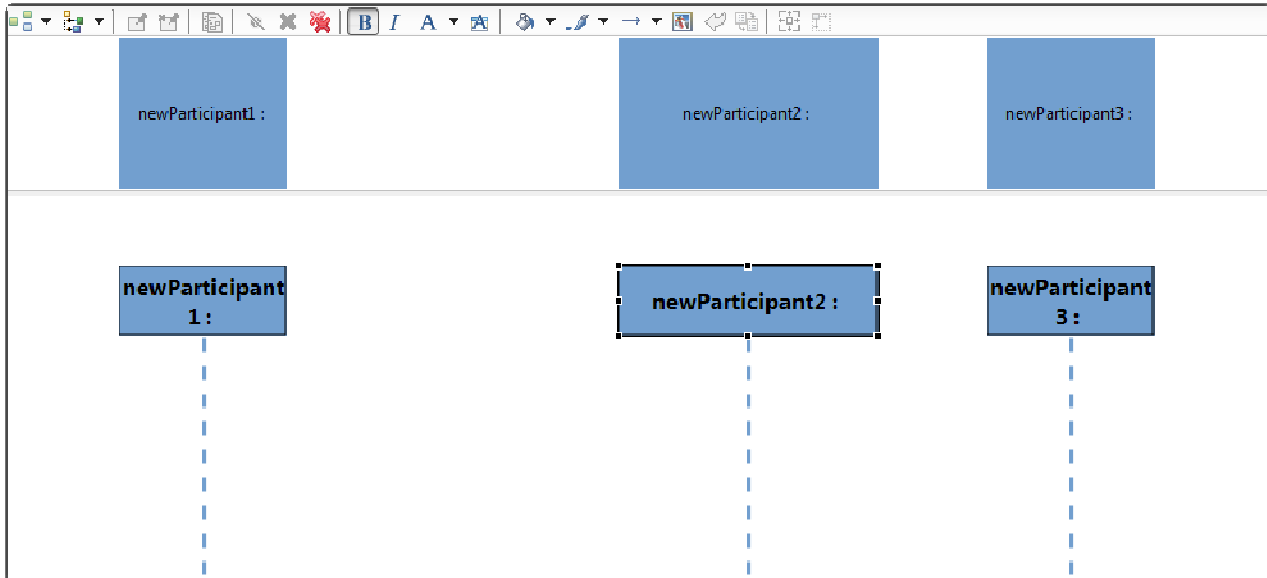
This header contains the objects corresponding to each lifeline. It is always visible, even when you scroll to the bottom of the sequence diagram.



An extension point is now available if you want to contribute this kind of header to your own diagrams.

Resize of lifeline header in sequence diagrams:

The lifeline header of the Sequence Diagram can be resized by the user to display the full label of the object corresponding to each lifeline.

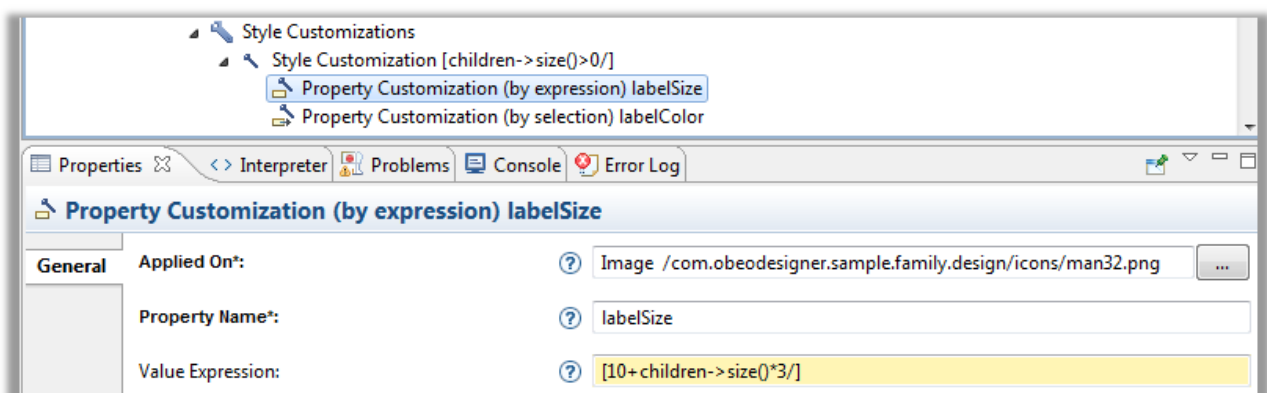
**Large models management with the Model Explorer view:**

The Model Explorer view has been improved to facilitate the management of large models, (containing objects with several hundreds of sub-objects). Now, the sub-objects are displayed in groups to facilitate and to accelerate the navigation through large models.

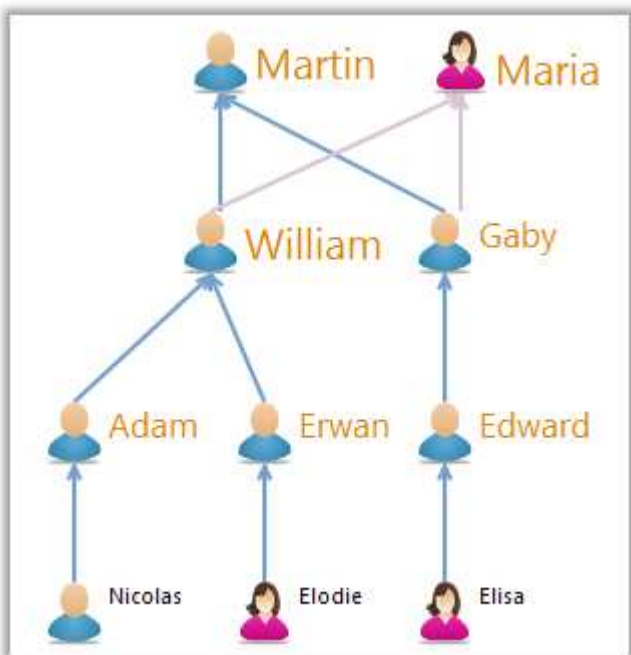
Conditional graphical properties

Graphical styles can now be customized property-by-property (label size, icon path, border color, etc). It means that you can link only some properties to a condition, the other properties will be automatically inherited from the default style.

For example, a condition can be defined in order to change only the color and the size of a label.



Here is the result of this customization: the size and the color have changed only on persons with children.



With this new feature, you can more easily combine conditional styles and reuse them on different objects.

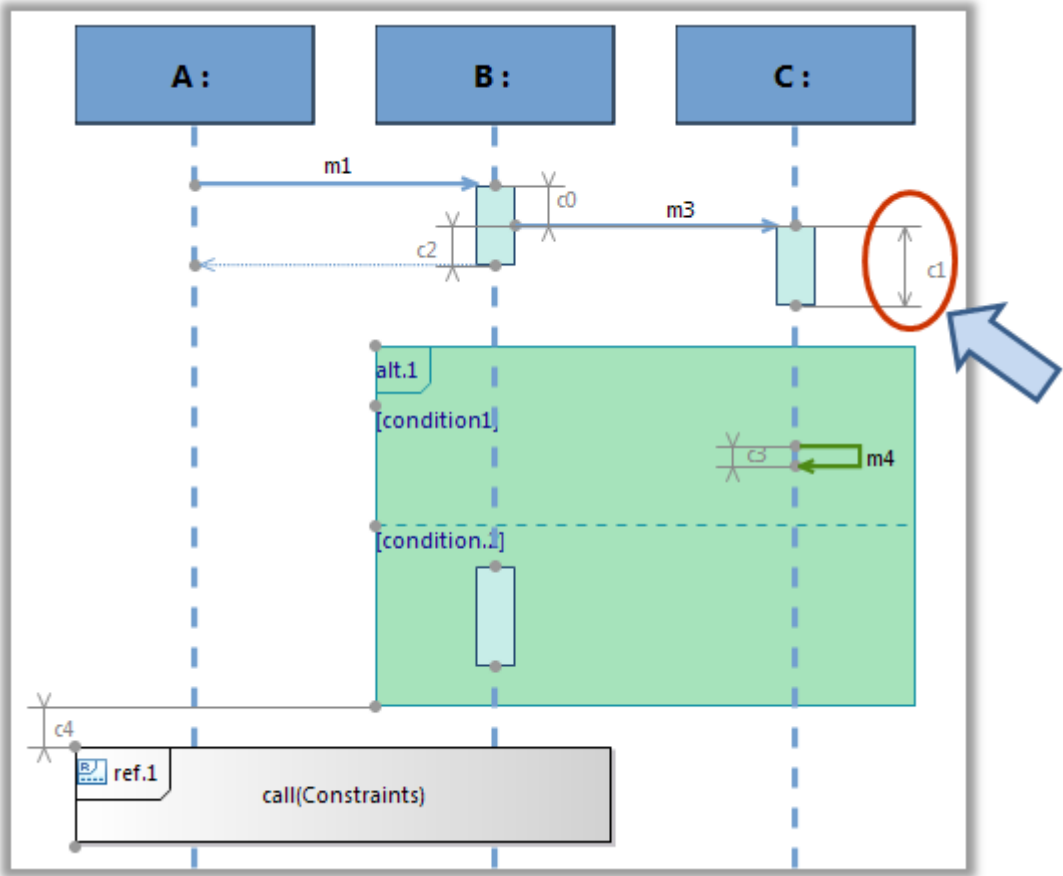
Style Customization scalability:

You can use regular expression in Viewpoint URI and Representation Name fields. This allows to extend several diagrams with one single diagram extension, which simplify the application of customizations on a set of elements. These fields are considered as regular expressions if they contain at least one of these characters: "*", "[", "]", "(", ")", or "?".

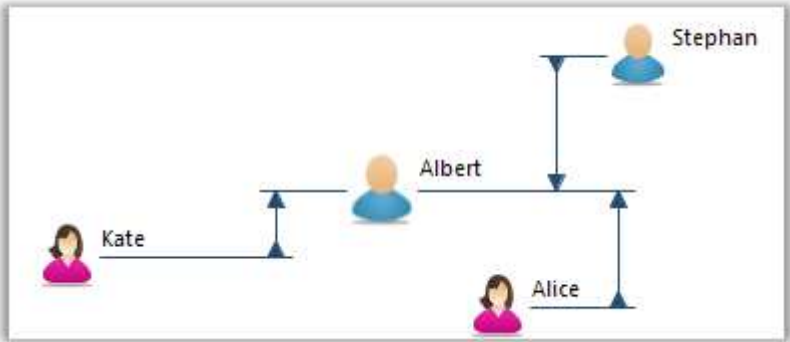
Bracket Edge style:

Bracket Edge is a new style for relations.

It allows you to represent relations, such as a measure, in the form of, for example, a vertical arrow between two segments.



It can be used on any edge of your diagrams. You can customize the color, the style, the thickness and the ends of the bracket.

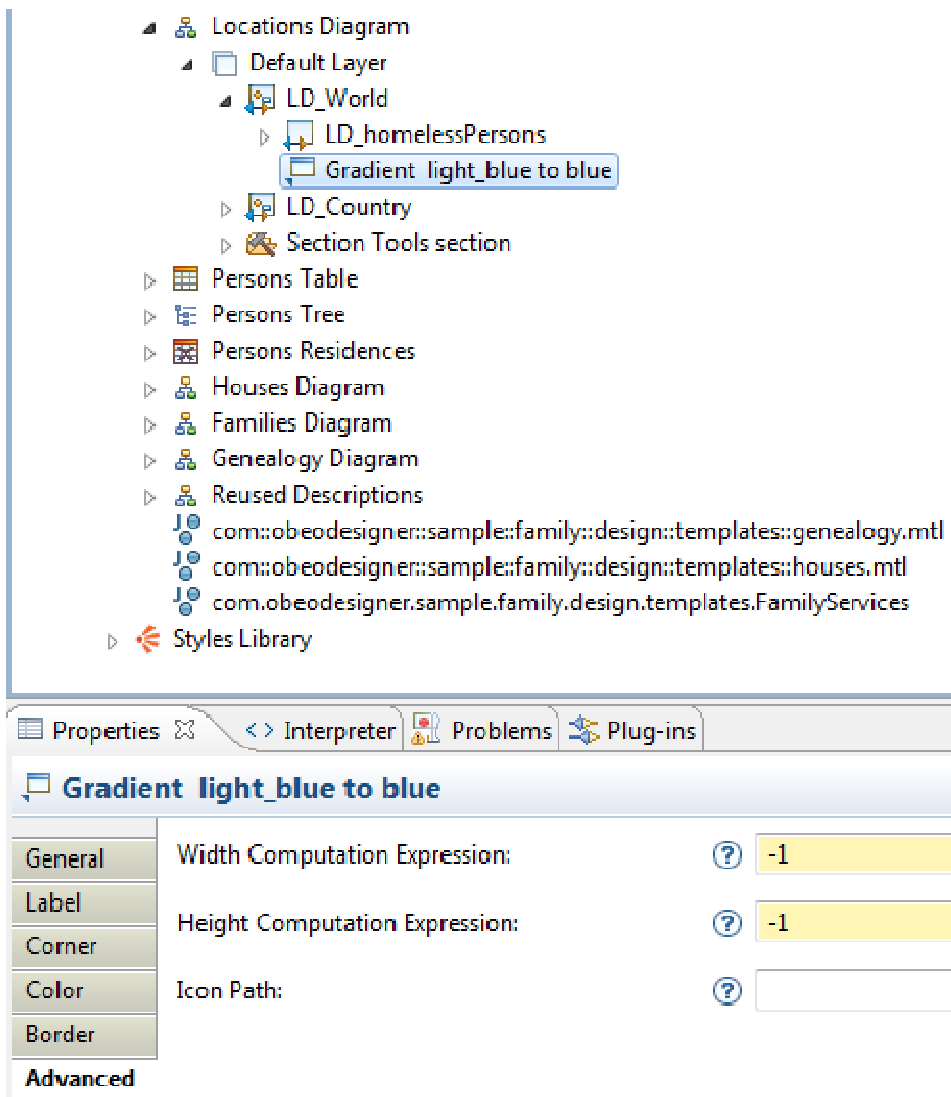


Additional layers:

Additional layers (formerly Optional layer) cannot be disabled by end-user if the specifier (person in charge of using Sirius to build a diagram editor) unchecked the "optional" option. In this case, the layers become mandatory.

Default size of containers:

Now, with Sirius, you can set the default size of the containers, as shown in the figure below:



It is very useful to directly create new containers with a more appropriate size, thus avoiding users to systematically resize them after their creation.

More progress bars to improve the user experience:

Many progress bars have been added for potentially long lasting actions (for example, opening large models).

Others fonctionnal/non-fonctionnal improvements:

A number of other improvements have been realized :

- Drag & drop of an element drops it at the mouse position in a container
- « Hide label by default » now works on Bordered nodes
- Select a default routing style for all new edges in end user's preferences, and set rectilinear style as default

- The property view is now synchronized with the Model Explorer view on semantic element selection
- Better layout engine
- Add tool variables definitions in the documentation
- Tools can't be created outside a layer anymore
- All tools now refresh the tree
- Better support of tabbar in Eclipse 4.x
- Fixes for Bugs and performance issues

New Branding (related to dissemination activities) :

A certain amount of tasks and actions have been done in order to communicate around Sirius outside of Crystal. These actions are not technical but are worthwhile to know for the Crystal project :

- New logo
- New website (<http://eclipse.org/sirius/>)
- Communication materials
- 4 related talks at Eclipse con Europe in Stuttgart (October 2013)
- 4 related talks at Eclipse con America in San Francisco (March 2014)
- An Eclipse Foundation newsletter dedicated to Sirius
- A very active forum (~20 message/week)

Link to internal working documents:

TI NAME: Sirius - Ecoretools 2					
TI_ID	TI_0032	Kind of TI	T	Contact email	Stephane.lacrampe@obeo.fr
Description: This task consists of a complete re-architecture and rewriting of the EcoreTools Eclipse project so that it is based on Sirius technology. EcoreTools is quite a central project for all the Eclipse Modeling Framework community (EMF) since this modeler is used by people who want to design their own meta-models. So this should bring quite a lot of visibility to Sirius.					
Link to internal working documents:					

3.4 Integration and Interoperability

TI NAME: Sirius - Modularization					
TI_ID	TI_0031	Kind of TI	T	Contact email	Stephane.lacrampe@obeo.fr
Description: TI_0018 introduced a major API break. It creates a technical opportunity to rethink the architecture of the technical component by introducing much more modularization. This modularization is driven by Business needs. On one hand, this is fundamental for this component to be able to achieve better integration within the context of Crystal, on the other hand, this is also critical to be able to address new client needs such as Web access to modeling information and edition.					

To achieve this properly, the modularity of the new architecture has been defined according to different analysis criteria :

- Common vs Runtime vs Tooling
 - o Runtime code is needed to execute a deployed Sirius modeler;
 - o Tooling code is needed to develop a Sirius modeler;
 - o Common code is shared by both runtime and tooling;
 - o Both runtime and tooling plug-ins can depend on common plug-ins (that is their purpose). Tooling plug-ins may depend on common and runtime plug-ins (for example to make sure the tooling knows exactly how something will be interpreted by the runtime), as long as the runtime plug-in is not more dialect or technology-specific than the tooling plug-in (for example, a generic tooling plug-in cannot depend on a diagram-specific runtime plug-in).
 - o Generic vs Dialect-Specific vs Technology-Specific
- Generic means dialect-agnostic, it corresponds to code which is common to all kinds of representations supported by Sirius (diagrams, tables, etc.)
 - o Dialect-Specific corresponds to the core semantics of a kind of representation (e.g. diagrams), independently of the concrete technology used to implement it;
 - o Technology-Specific corresponds to the concrete implementation of a dialect on top of a specific technology (e.g. GMF Runtime for diagrams).
- Core vs Eclipse-specific vs UI
 - o Core code is pure library code with no dependency on any runtime framework (or maybe only OSGi), and can be run headless and outside of an Eclipse runtime;
 - o Eclipse-Specific code depends on the Eclipse framework, but not on the UI; it can still be run headless;
 - o UI code requires a complete Eclipse Workbench (and provides the integration inside of it).
- API vs internal API vs SPI vs internal (this is more for package boundaries than plug-in boundaries)
 - o API is the official API that can be used by non-Sirius code;
 - o Internal API is technically API (i.e. exported by the OSGi bundles) but reserved for internal usages inside of Sirius itself. It can be used by non-Sirius code but it does not have the same kind of guarantees in terms of backward compatibility;
 - o SPI (Service Provider Interface) is specifically targeted for systems which want to extend Sirius. This typically includes extension points and interface definitions that extender must implement. These may be used by Sirius itself, for example to provide default implementations.
 - o Internal is all the rest, and is not accessible outside of Sirius.

Note that these criteria should not dictate the final architecture, but simply allow a better control and understanding of the dependencies. Simply creating a bundle for each combination of these criterion and put each existing piece of code in the correct plug-in bucket would not result in any kind of meaningful architecture. However, the existing code base is more or less organized along such semi-technical boundaries and a good first step would be to ensure this structure is actually correctly enforced.

It is also important to note that this modularization is done in such a manner that the behavior of Sirius is not deteriorated in any way in the end.

This architecture big blocks are :

- Library extensions
- Sirius Core Platform and Services
- Sirius Diagrams
- Sirius Sequence Diagrams
- Sirius Tables
- Sirius Trees

More information can be found at : <http://wiki.eclipse.org/Sirius/Modularization>

For information, at M+12, this work is still ongoing.

**Link to internal working
documents:**

4 Open Source component for generating GUI presentation of business data brick

4.1 Description

Name:	Yves YANG
Contact:	yves.yang@soyatec.com +33 6 20 74 39 45
Technical information:	This brick is a component of industrial maturity for the development and to generate application UI for data presentation. The component allows to define the UI logic in a neutral way first, and then to generate the native codes for each target device or display environment. It targets to be a very productive development component for UI for enterprise software engineering.
Dependencies	This brick is based on the Eclipse platform and depends on Eclipse components like EMF, EGF and XWT.
License	This brick is provided as an open source component under the Eclipse Platform License (EPL V1.0 - http://www.eclipse.org/legal/epl-v10.html).
Additional information	The brick specification can be found on the AVL Sharepoint for the Crystal project : https://projects.avl.com/11/0154/Data%20Exchange/007_Work/SP6_RnT_Activities/6.9%20Multi-viewpoint%20Engineering/PMF_V_1.2-CRYSTAL.PDF

PMF is not a specific UI framework of displaying technology, neither a specific UI Markup language. It is a UI technology, data model independent and highly extensible MDE framework to design and produce business data presentation, scenario/pattern and user interactions.



PMF offers a common way to specify a UI independently of any programming language & infrastructure used for the final application. The framework allows to describe the characteristics of a user interface in different abstraction levels, and then using the Model-Driven Engineering (MDE) approach/paradigm to generate the executable UI.

The development process can be summarized as follows:

- PIM (Platform Independent Model) logic design;
- PSM (Platform Specific Model) design;

Version	Nature	Date	Page
V01.00	R	2014-04-2830	28 of 117

- Transformations/Generators to produce the executable UI for the targeted specific UI technology platforms.

This framework presents a lot of important benefits on productivity over the traditional programming solution:

1. Separation of each developer role to application developer and UI component developer;
2. Keep the entire application system consistent in terms of software architecture;
3. Easy to evaluate;
4. UI device and technology independent.

4.1.1 Manual

This component is still in development phase. The result is not yet published in public community of eclipse. The release of this brick is available only for the partners of this project. It can be hosted on Soyatec Website. It can be installed directly from eclipse Update by using this link <http://www.soyatec.org/update/pmf>. The specification of meta-model can be downloaded from this link: http://www.soyatec.org/pmf/PMF_V_1.2.pdf.

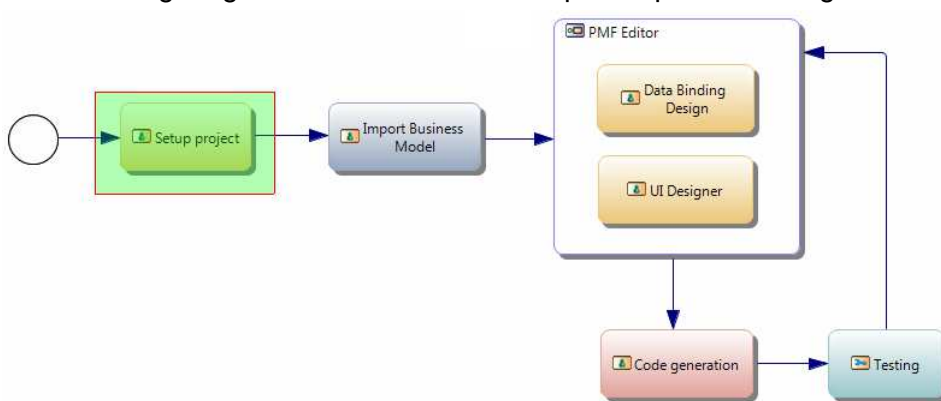
This component is developed and tested in this following environments

Eclipse	4.2+
Java	1.6+
Operation System	Windows XP/7/8 32/64 bits, Linux : Ubuntu, RedHat MacOS 10.6+

This page indicates the steps to install PMF in eclipse:

<http://www.jiraproject.com/confluence/display/PMF/Installation>

The following diagram illustrates the development process using PMF:



Phase	Name	Description
1	Setup Project	Create a resource file with initial system type and data. This file will be used for the UI design.
2	Import Business Model	Import the business data model that will be presented in UI.

3	PMF Editor	UI Design in PMF.
4	Code Generation	Launch the code generation.
5	Testing	Run the application and test it.

To get started with the development of PMF, you can follow [this tutorial to generate “Properties View” for the model Library](http://www.soyatec.org/pmf/PMF-01.html) which is used as demonstration of PMF:

<http://www.soyatec.org/pmf/PMF-01.html>

4.2 Use Case coverage and application

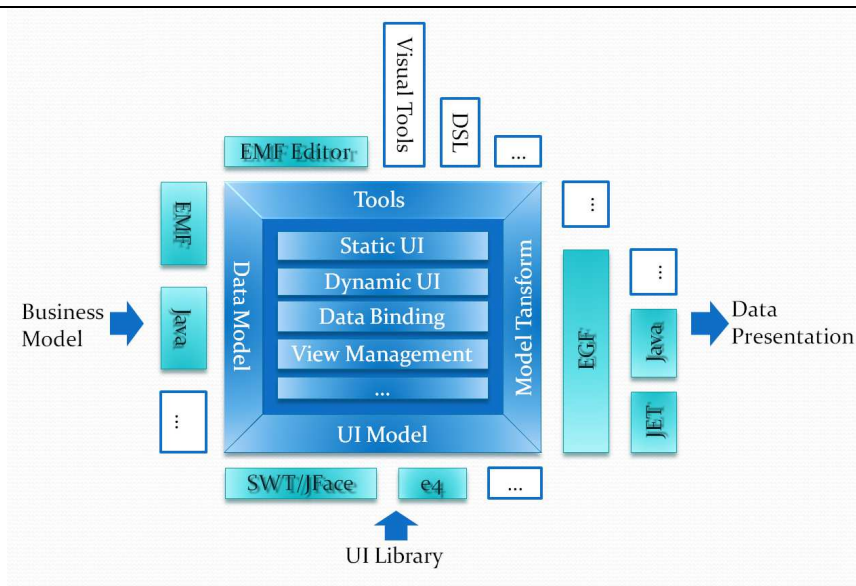
This meta-model of component is platform independent (PIM). It provides a capability of abstract UI modeling for different Data Models, UI display device or technology and programming languages. Therefore, it can be integrated into any modelling solutions developed in Eclipse such as Sirius which is developed in the context of this project, or commercial products like [IBM Rational Software Architect](#) or [Modeller](#).

There are several important aspects to take into consideration in the design of this framework:

1. High extensibility in each level such as UI, code generator etc.;
2. Support of most used data models (note : data models can be mixed together in a same application);
3. Possibility to integrate with most used UI technologies; different UI technologies can work in the same application;
4. Easy to use :
 - a. Integrated with the current traditional development environment;
 - b. Possibility to adopt this new approach gracefully;
 - c. Reuse of the existing UI component or adding a new ones;
 - d. Possibility to extend the code generator;
 - e. Keeping all concepts simple.

4.3 General Improvement

TI NAME: PMF – Implementation of meta-model					
TI_ID	TI_0043	Kind of TI	MM	Contact email	yves.yang@soyatec.com
Description: The meta-model is the key module of PMF, which provides the capability to design the application UI in an abstract model. The UI development of enterprise applications consists of the presentation modeling using the model provided by PMF in an abstract way. The abstraction concerns both Data Model and UI Widget model. The Data Model defined in PMF is a lightweight model that summarizes all possible data models designed only for data binding of UI Presentation. Data Binding is a powerful concept and solution to associate the Data model with UI display widgets.					



The UI Model is a lightweight model that encapsulates the common structure and behavior of all UI Widgets such as Swing, SWT/JFace, HTML, Ext etc.

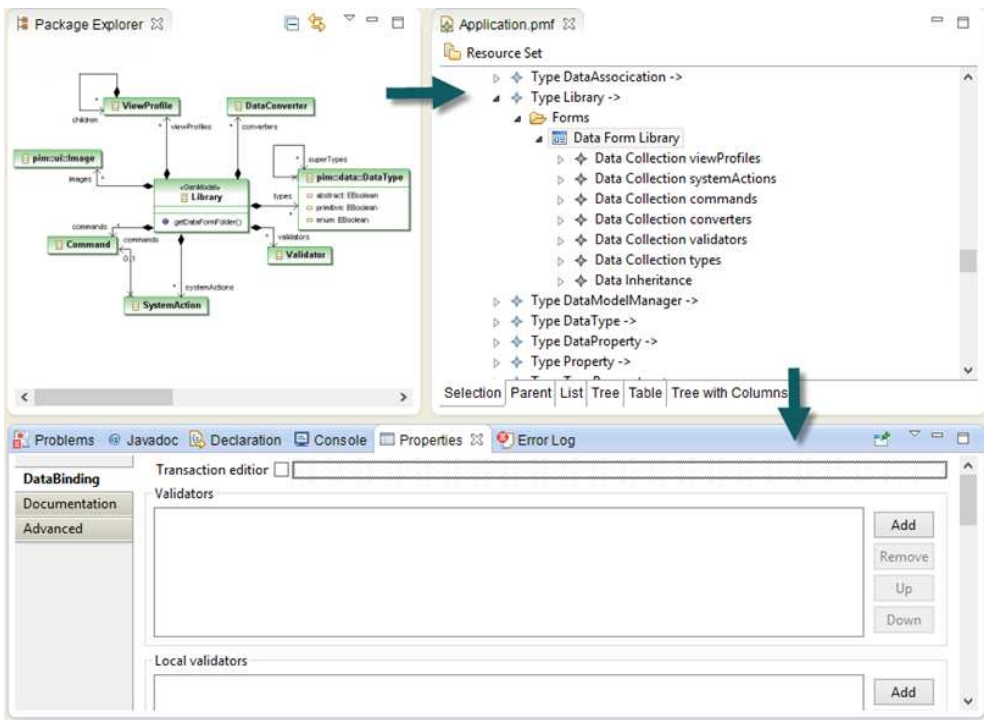
The UI modeling is divided into different phases:

- Static UI
- Dynamic UI
- View Management

The Static UI deals with which data is accessible via UI. The Dynamic UI deals with when this data is accessible, i.e. accessibility of the data on all kinds of events such as UI event, Data Modification, etc. The View management gives the possibility to define different presentation for each data type. Many flexible mechanisms can be used to select one of the defined data presentations. More advanced presentation patterns can be used directly in UI design such as Master/Detail, Wizard, Finder, etc.

Link to internal working documents:

TI NAME: PMF – Integration with EGF					
TI_ID	TI_0044	Kind of TI	T	Contact email	yves.yang@soyatec.com
Description: The purpose of this integration is to provide a highly extensible and flexible code generation infrastructure based on EGF. This integration allows the developers of PMF to extend the code generators not only by using all supported template code engines such as JET or Acceleo, but also by any programming languages such as Java, Ant or JRuby.					
Link to internal working documents:					

TI NAME: PMF – Generation of Properties View					
TI_ID	TI_0045	Kind of TI	T	Contact email	yves.yang@soyatec.com
<p>Description:</p> <p>As a modeling engineering component, PMF provides a set of tools to ease the developer's life. The Properties View for the PMF model is one of most used view parts in eclipse for the model edition.</p> <p>This module is in fact the first application of PMF. The Properties View is modeled in PMF meta-model and generated by the integrations with EGF and XWT.</p> 					
<p>The following features are supported:</p> <ul style="list-style-type: none"> • Multiple Tabs • Documentation in Rich text • Undo/Redo • Support of any editors with editing Domain • Collection, List and Table with the modifications Add, Remove and Move up and Move down <p>This integration is limited only in Static UI. Further integration will be done in the next release.</p>					
Link to internal working documents:					

TI NAME: PMF – Integration with XWT					
TI_ID	TI_0046	Kind of TI	T	Contact email	yves.yang@soyatec.com
Description: <p>XWT is a declarative UI in XML for eclipse. It respects the Open standard XAML, defined by Microsoft. It is a perfect runtime solution of model driving system. PMF relies on it for the development of tools for PMF model edition.</p> <p>A Library is defined to host all the necessary logic resources for the code generation for application in XWT. The resources include converters and validators. The associated implementation resources are declared for the code generation. A set of template patterns can be used directly for the UI generation in XWT.</p> <p>Precisely, this integration includes the code generations of all EMF types and most used UI types such as Text, CheckBox, Combox, etc.</p> <p>This integration is limited only in Static UI.</p>					
Link to internal working documents:					

4.4 Integration and Interoperability

The main task of this brick is to develop the core of PMF. The integration with other solutions and the interoperability are not part of the brick scope for the moment.

5 Open-Source Components for Model Transformation, Model-Driven Code Generation and Model validation and analysis brick

5.1 Description

Name:	Benoît Langlois
Contact:	benoit.langlois@thalesgroup.com - +33 1 70 28 23 53
Technical Information:	This section presents three bricks dedicated to: 1) model transformation, 2) model-driven code generation, and 3) Validation of models.
Dependencies	Those three bricks are based on the Eclipse platform and depend on Eclipse components such as EMF. They are integrated in the Kitapha development and execution environments.
License	This brick is provided as an open source component under the Eclipse Platform License (EPL V1.0 - http://www.eclipse.org/legal/epl-v10.html).
Additional information	This brick will be Open Source in the Kitapha Eclipse project. Kitapha website: http://www.eclipse.org/kitapha/ - this website will be created in April/May 2014.

Model Transformation

Model transformation is a key brick in building model-driven engineering environments. It allows implementing tools for transition from one level of abstraction to another, from a component model to another one, or from a general formalism to a formalism (e.g., DSL to UML, MOF to UML) of a tool dedicated to a particular concern (e.g. performance, safety, or security analysis). Historically, model transformation was mainly used for "PIM to PSM" (Platform Independent Model to Platform Specific Model) transitions but it is more used as a tool interoperability and synchronization now.

There are several conversion technologies dedicated to technologies Eclipse (EMF) models. These technologies commonly come with a dedicated description of transformations (ATL, QVT...) language.

The state of the art provides many tools and technologies but these are strictly dedicated to model transformation. Now, many applications around models (model transformation or information structured according to a grammar / meta-model, optimized code generation, various treatments from models such as the impact analysis or extracting parts models) face common problems with the transformation of models:

- Expression of tasks in the form of independent rules;
- Grouping sets of rules;
- Inference rules (mutual exclusion);
- Scheduling rules (application of right rules for the right concerns, in the right order);
- Expansion / redefinition of rules or sets of rules (specialization, overload, capitalization).

The languages available are languages formalization and do not have the same expressiveness than conventional implementation languages (Java), and are dedicated to the specific problem of

Version	Nature	Date	Page
V01.00	R	2014-04-2830	34 of 117

model transformation. This makes languages dedicated to specialists and a risk of dispersion of practices, i.e. the dispersion to multiple existing languages at a low TRL.

This component provides a technical framework for model transformations allowing the definition of rules in Java, rich and widespread language, and the definition of rules for any type of information (model, XML...), headed by a grammar (XML, meta- model... diagram). It must also provide the ability to declare the rules by "sets" which are clusters of rules working in the same treatment and are compatible with a given format. It should not be dedicated to Eclipse / EMF, but should be quite generic.

The "sets" of rules must provide the ability to be extended, specialized, or reused. The objective is that a given model transformation must be reused and even specialized in order to process further information or other kind of transform later.

This component is rule-oriented and provides two mechanisms:

- Inference: for every element involved in a transformation, a rule declares premises in order to determine a precedence order between all the transformation rules. For instance, a super-class must be transformed before its sub-classes.
- Scheduling: from all the declaration of inference, a scheduler deduces the order to apply all the transformation rules.

The formalism is declarative. Inference and scheduling dramatically reduces the effort of the developer to create transformations, incrementally add rules, or change a transformation order. Then, the complexity of transformation is reduced and transformations are better managed.

In the remaining of the document, the brick to support this environment is called Transposer.

Model-driven code generation

In model-driven engineering, and more generally in any field in which from some formalized representation of an object or set of objects is produced in a range of information a set of files, build tools, or "generators" are required.

Several technologies are currently available and widely used for the production of code generators: Acceleo XPand, StringTemplat... These technologies are similar in that they provide languages (standard or otherwise) for expressing generation patterns ("templates"). They provide further assisted development environments. However, these technologies only address a sub-part of the problems of code generation.

It may appear that in the life cycle of a generator for a single representation of an object or set of objects, the generated product changes of form, organization, e.g. in terms of directory structure, file name, number of files, distribution of information in the files to satisfy external constraints. Similarly, the syntax information to generate the files may also vary, for example to take into account a change of standard.

The combination of changes in form and content can lead to situations of peril for maintenance and scalability of a generator. A technical environment for the generation from models should define how to avoid this risk by separating the form management from the content management in the architecture of a generator.

Traditionally, the problem of generation is solved in two ways:

- Handling a program representation to generate files. In this approach, the code generator contains manipulations of the representation of files to generate. From the developer point of view, this approach seems simple because the generator is a unique program, written in a single language. The major drawback of this approach is that the resulting generator combines different aspects in a monolithic architecture that will be difficult to maintain.

Version	Nature	Date	Page
V01.00	R	2014-04-2830	35 of 117

- Application patterns (templates) using specific languages, for example Acceleo, Xpand, String-Template, Velocity. In this approach, the use of patterns makes it easy to describe the content of different types of files to produce. The part of the generator that creates files is written in a program and the patterns are expressed in specific languages.

When the generated product is code written in a programming language, it is necessary to produce compiler directives that link files and content between them, for example, and according to the programming language : # include, with, import, use.

The computation of these directives is strongly linked to the organization of produced files, which introduces a strong coupling between the program that implements the generation patterns and these patterns themselves. It is therefore impossible to change the organization of files produced without changing the program that applies patterns and generation patterns themselves.

Maintaining consistency in the program and patterns is a major drawback, which includes the risk of these two sets to derive and become inconsistent and hardly maintainable.

The main innovation of this brick is a technical framework providing a modular architecture for code generators, separating the issues of generation, while providing the freedom to choose or compose technologies already present in the open-source community.

In the remaining of the document, the brick to support this environment is called Composer.

Validation of models

Model validation is a core function in modelling which ensures that a model is coherent, complete and respects domain constraints. When a model is valid, it is next possible to apply operations such as model transformation or code generation; otherwise those operations may fail. Most of modelling tools verify that a model conforms to its meta-model; however additional constraints must be added in order to address a specific domain.

In the Eclipse environment, model validation is by default provided by EMF, which detects for instance cardinality errors. Nevertheless, in order to address an engineering domain or context, validation must be enriched by specific validation constraints. Every constraint is an invariant that must always be respected.

EMF provides an extensible validation component, named “EMF Validation”, with a constraint-based evaluation service over models. Constraints can be written in various languages, such as Java or OCL. OCL is a constraint language defined by the OMG. If EMF Validation is efficient to ensure model validity, it is not accessible to an EMF non-expert and deserves ergonomics and usability evolutions.

The purpose is to provide an environment which, from an ergonomics viewpoint, facilitates the definition of constraints, either for tool developers or end-users by hiding the complexity of EMF Validation and unifying the ways to declare them. From a modularity viewpoint, constraints can be grouped by category in order to activate/deactivate pools of constraints.

In the remaining of the document, the brick to support this environment is called Accuracy.

The specification of this brick was written in the context of the French Sys2Soft project. It can be found on the AVL Sharepoint for the Crystal project:

https://projects.avl.com/11/0154/Data%20Exchange/007_Work/SP6_RnT_Activities/6.9%20Multi-viewpoint%20Engineering/Model%20Validation%20and%20Analysis-1.0.0.docx

Version	Nature	Date	Page
V01.00	R	2014-04-2830	36 of 117

5.1.1 Manual

This part will be written when the Eclipse component will be created and the web site created.

5.2 Use Case coverage and application

Report to the Sys2Soft specification for Model validation:

https://projects.avl.com/11/0154/Data%20Exchange/007_Work/SP6_RnT_Activities/6.9%20Multi-viewpoint%20Engineering/Model%20Validation%20and%20Analysis-1.0.0.docx

5.3 General Improvement

Model Transformation

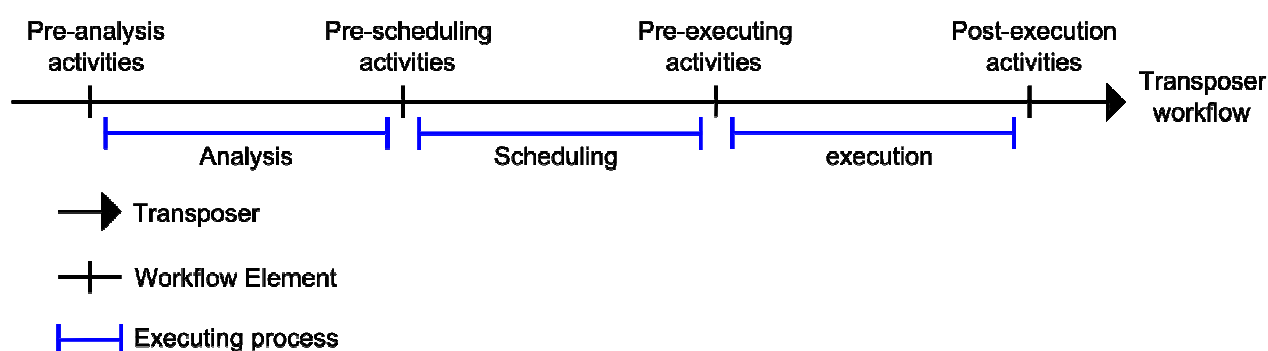
TI NAME: Specification of Transposer					
TI_ID	TI_0097	Kind of TI	T	Contact email	benoit.langlois@thalesgroup.com
Description: <p>Transposer is a component that allows applying actions on a source model for a given purpose. Model transformation is the main purpose but it can be extended to other ones such as migration, import/export, or incremental transformation when traceability is introduced. The stake of Transposer for developers is to reduce the complexity of model transformation and to improve the management of model transformations.</p> <p><i>The structure of declaration of a model transformation</i></p> <p>A transformation is defined by a mapping decomposed in a set of mapping elements which contains mapping rules. The structure is the following:</p> <ul style="list-style-type: none"> • Mapping: it declares a transformation mapping <ul style="list-style-type: none"> ○ Mapping purpose ○ Mapping name ○ Description ○ Domain: declaration of the source metaclasses concerned by the mapping ○ Context: global context of the mapping ○ Mapping Elements: decomposition of the mapping in a set of elements, each describing a metaclass mapping <ul style="list-style-type: none"> ▪ Name ▪ Domain metaclass: source metaclass to be mapped ▪ Mapping rules: the set of rules which map the source metaclass into target metaclasses <ul style="list-style-type: none"> • Name • Rule: Java implementation rule • Context: local rule context • There is a distinction when there is a cycle to map a source metaclass (e.g., Element contains a set of Element) <p>A Java rule contains the following information:</p> <ul style="list-style-type: none"> • A set of premises to apply the rule: it defines the precedence order to map a source metaclass. For instance, a Class must be mapped in the target location before its Attributes and Operations. • The Apply action to map a source metaclass into target metaclasses. 					

The mechanisms of Inference and Scheduling

A Java rule declares premises in order to manage constraints of precedence order between source metaclasses to be mapped. For instance, a rule for Attribute mapping declares that the Class metaclass must be mapped before the Attribute metaclass. The Inference function computes all the mapping dependencies. From this computation, the scheduler schedules the order to apply the mapping rules.

The model transformation workflow

The following picture shows the complete workflow of model transformation.



Description of every part of the workflow:

- The Analysis corresponds to the Inference function
- The Scheduling corresponds to the scheduling function
- The Execution phase corresponds to the application of the mapping rules for model transformation
- The “Pre-analysis activities” step enables to execute activities before the Analysis step
- The “Pre-scheduling activities” step enables to execute activities before the Scheduling step
- The “Pre-executing activities” step enables to execute activities before the Execution step
- The “Post-executing activities” step enables to execute activities after the Execution step

Link to internal working documents:

TI NAME: Prototype of Transposer					
TI_ID	TI_0098	Kind of TI	T	Contact email	benoit.langlois@thalesgroup.com
Description: The work achieved in April 2014: <ul style="list-style-type: none"> • Implementation of Transposer: <ul style="list-style-type: none"> ○ Implementation of the structure to declare a transformation mapping ○ Implementation of the mechanisms of inference and scheduling to schedule mapping rules ○ Implementation of the workflow to apply scheduled mapping rules ○ Implementation of the mechanism of contribution to the workflow 					

- Realization of examples:
 - A “UML to ecore” model transformation
 - A “Simple Component” to “Viewpoint DSL” transformation in order to declare a Component viewpoint from a Simple Component model

The Innovations are:

- The inference and scheduling mechanisms
- Transposer is metamodel-agnostic
- The ability to insert activities in the transformation workflow
- The ability to address incremental transformations

The issues are:

- Complexity of implementation of the inference and scheduling mechanisms
- Complexity of implementation of the workflow extension with pre- and post-activities
- Adoption by the modeling community

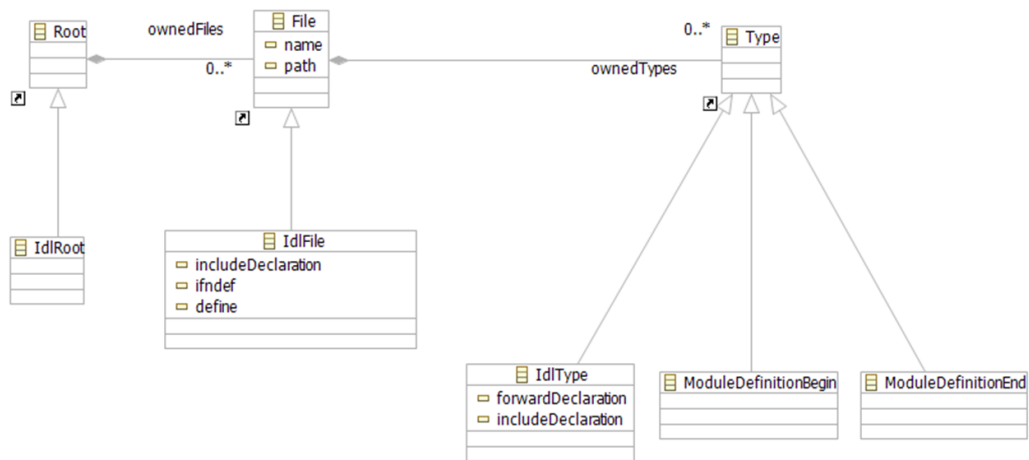
The results are:

- Availability of Transposer
- Availability of a component, named Cadence, common to Transposer (for model transformation) and Composer (for model-driven code generation) for the workflow implementation
- Availability of examples

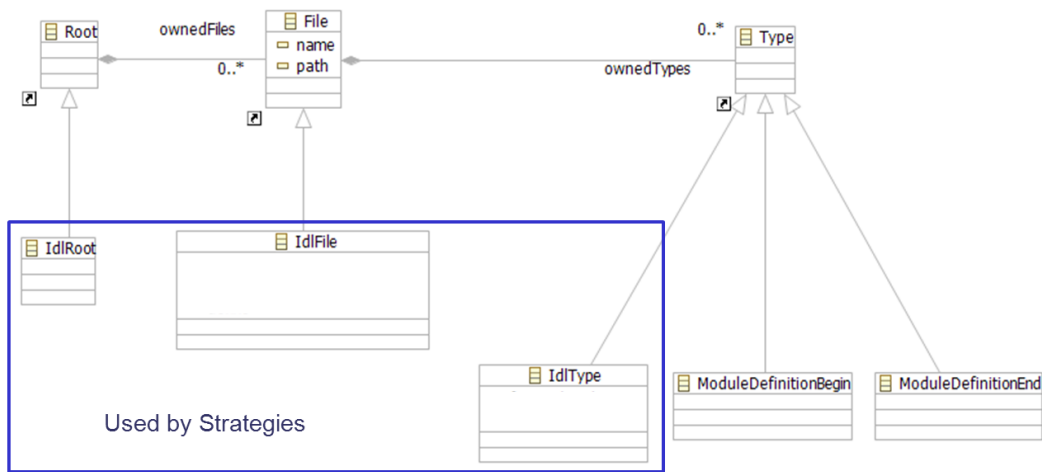
Link to internal working documents:

Model-driven code generation

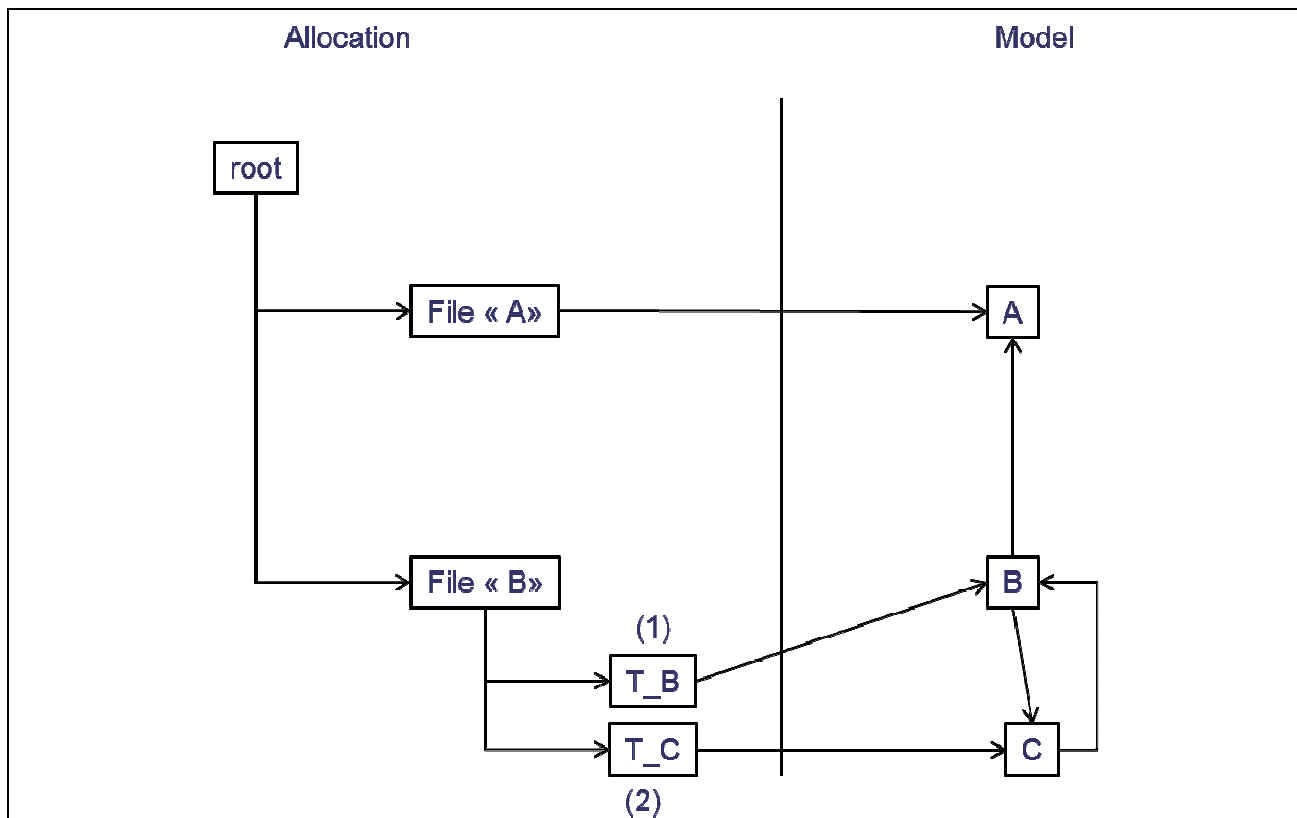
TI NAME: Specification of Composer					
TI_ID	TI_0099	Kind of TI	T	Contact email	benoit.langlois@thalesgroup.com
Description: <p>Model-to-text transformation, i.e. code generation from model, is a mature technology with tools such as Acceleo or Xpand. However, there are limitations. For instance, there is a lack of flexibility when a template is written, there is no control on the organization of generated files, and it is difficult to adapt the generated code to specific contexts. The purpose of Composer, a tool dedicated to model-driven code generation, is to add flexibility to organize generated files, to be independent of generation technology, to separate templates from algorithms.</p> <p><i>1. Concept of Strategy</i></p> <p>An Allocation defines a set of files, each with their own structure, which are bound to a set of model elements. Technically, an allocation is a “Root” containing a set of “Files”, each referencing a set of “Types”. A file is a file where code is generated. A type is a structure to be included in a file (e.g., a begin/end section). A type is associated to a metamodel element.</p>					



A Strategy is an algorithm which produces an Allocation from one model and a set of model elements.



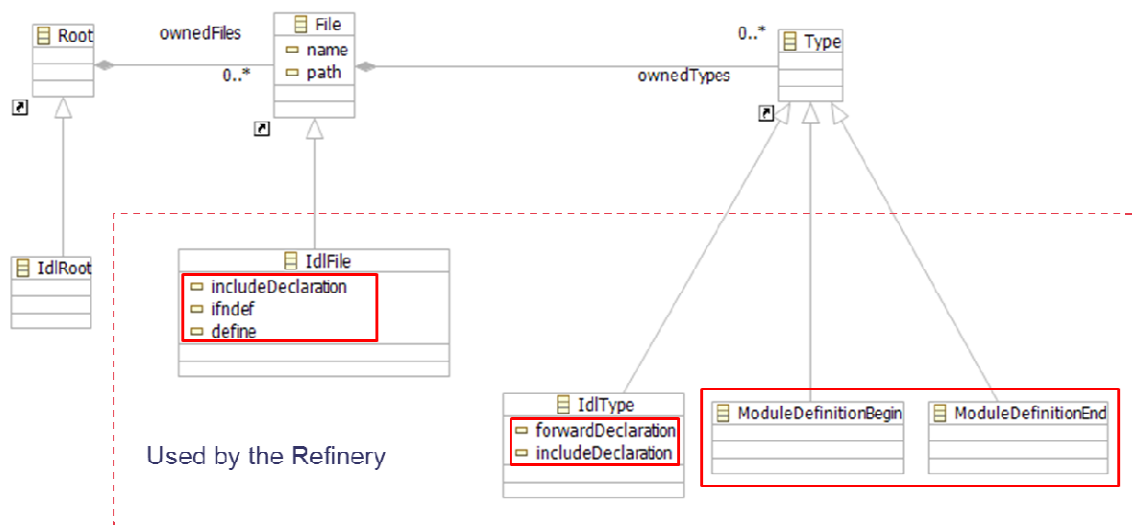
The following picture exemplifies an allocation for a set of model elements in different files with different code organizations.



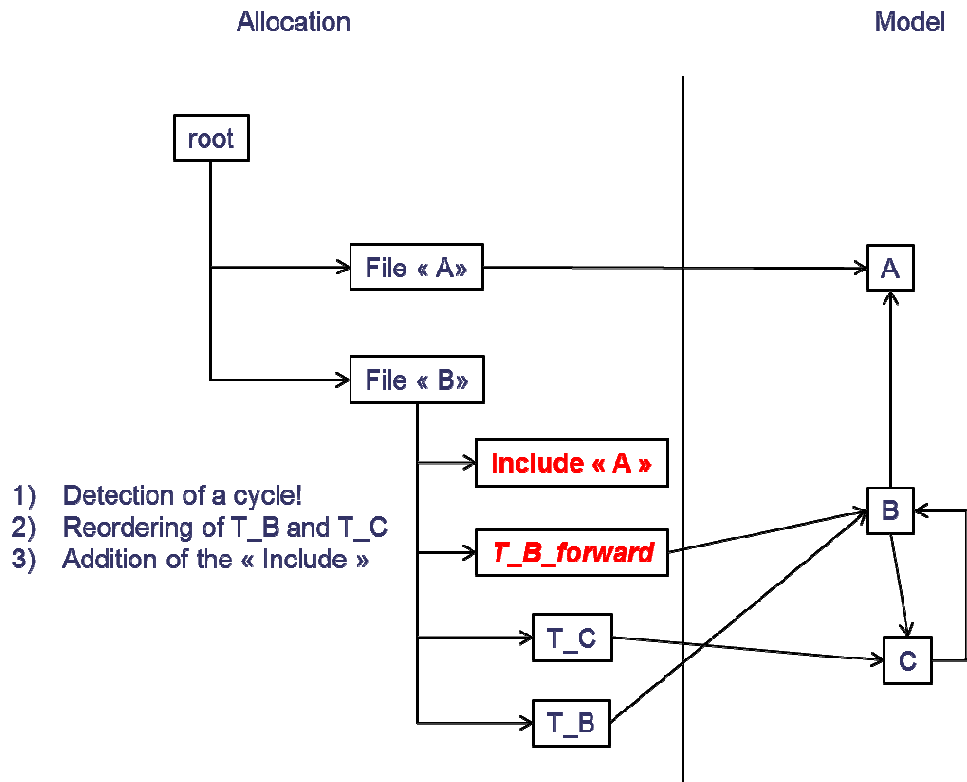
The interest is to select and configure a strategy, among several, which fits the need of file and code organization.

2. Concept of **Refinery**

The Refinery completes an Allocation. Typically, it re-orders the model elements in one file and identifies dependencies (e.g., imports, includes declarations).



The following picture exemplifies an allocation enriched by a Refinery.



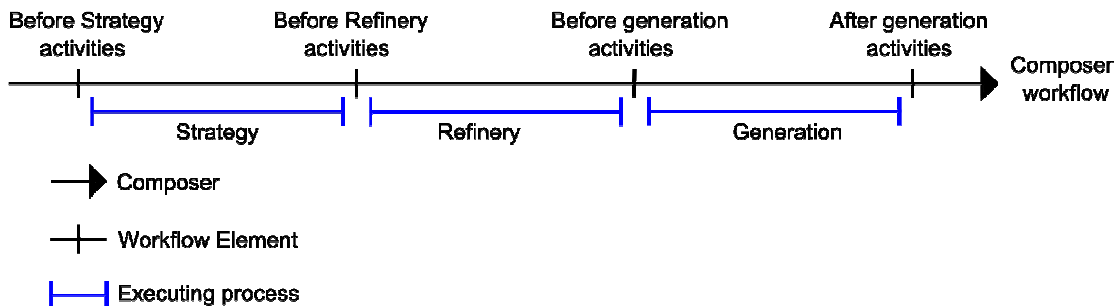
Whereas several strategies can be involved, there is only one Refinery to complete the code generation.

3. Concept of **Generator**

While the Allocations and Refinery set all the assembly details of code generation, a set of Generators (e.g., Acceleo, EGF, Jet, Java, StringTemplate) release the code generation itself. While Allocations and Refinery define a kind of generation plan, Generators generate code.

4. The Composer **workflow**

The following picture shows the complete workflow of Composer.



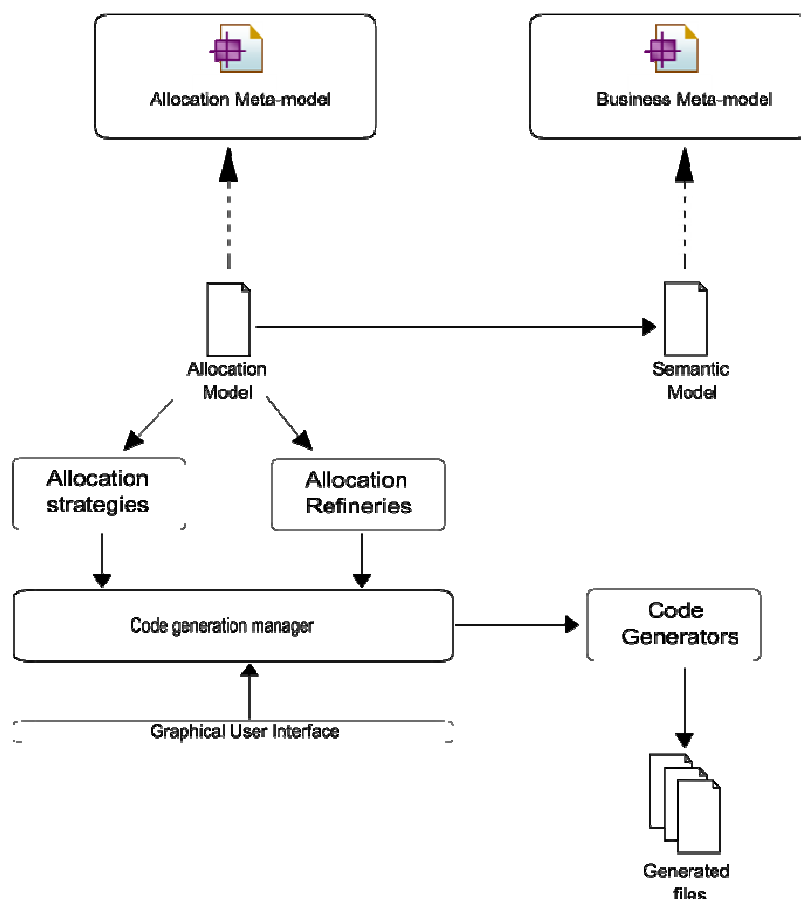
Description of every part of the workflow (different from the Transposer workflow):

- The Strategy defines an Allocation from several strategies
- The Refinery completes the work of Allocation

- The Generation applies the code generation from the generation plan previously defined
- The “Before Strategy activities” step enables to execute activities before the Strategy step (e.g., model validation, cleanup of the generation area)
- The “Before Refinery activities” step enables to execute activities before the Refinery step
- The “Before generation activities” step enables to execute activities before the Generation step
- The “After generation activities” step enables to execute activities after the Generation step (e.g., code formatting, commit code on a repository)

5. Code generation manager

The code generation manager is the component which orchestrates the code generation. Its code generation plan is defined by contributions of Allocation strategies and Refinery. A user interface enables to launch the code generation. At runtime, the code generation manager sets the Allocations, applies the generation with the support of code generators, and is enriched by contribution of activities between the step of Allocation, Refinery and Generation.



Link to internal working documents:

TI NAME: Prototype of Composer					
TI_ID	TI_0100	Kind of TI	T	Contact email	benoit.langlois@thalesgroup.com
Description: The work achieved in April 2014: <ul style="list-style-type: none"> Implementation of Composer: <ul style="list-style-type: none"> Implementation of the framework to declare Allocation strategies and Refinery for model to code generation Implementation of the code generation manager Implementation of the workflow to apply model to code generation Implementation of the mechanism of contribution to the workflow Realization of examples: <ul style="list-style-type: none"> An “ecore to html” generation with different strategies of generation A “Simple Component to html” generation The Innovations are: <ul style="list-style-type: none"> The dissociation of code template and algorithm The ability to generate in multiple files with their own structure The independence of code generator (e.g., Acceleo, Xpand, EGF) The ability to insert activities in the code generation workflow The issues are: <ul style="list-style-type: none"> Complexity of implementation of the complete allocation Complexity of implementation of the workflow extension with pre- and post-activities Adoption by the modeling community The results are: <ul style="list-style-type: none"> Availability of Composer Availability of a component, named Cadence, common to Transposer and Composer for the workflow implementation Availability of examples 					
Link to internal working documents:					

Validation of models

TI NAME: Specification of Accuracy					
TI_ID	TI_101	Kind of TI	T	Contact email	benoit.langlois@thalesgroup.com
Description: The specification is available on the AVL Sharepoint.					
Link to internal working documents:					

TI NAME: Prototype of Accuracy					
TI_ID	TI_102	Kind of TI	T	Contact email	benoit.langlois@thalesgroup.com
Description: The work achieved in April 2014: <ul style="list-style-type: none"> Implementation of Accuracy: <ul style="list-style-type: none"> Implementation of the Accuracy framework of validation with the possibility to edit OCL and Java constraints Execution of constraints based on Accuracy and EMF Validation Possibility to activate/deactivate groups of validation constraints Realization of examples: <ul style="list-style-type: none"> Declaration and application of constraints on the “Simple Component” metamodel The Innovations are: <ul style="list-style-type: none"> Ability to declare constraints at a high level of description Ability to declare constraints in a language – Java and OCL are supported today Ability to activate / deactivate groups of constraints The issues are: <ul style="list-style-type: none"> The high level descriptions could be based on tool such as Xtext in order to improve the constraint edition (e.g., for assistance, completion) The results are: <ul style="list-style-type: none"> Availability of Accuracy Availability of examples 					
Link to internal working documents:					

The specification of this brick was written in the context of the French Sys2Soft project. . It can be found on the AVL Sharepoint for the Crystal project:

https://projects.avl.com/11/0154/Data%20Exchange/007_Work/SP6_RnT_Activities/6.9%20Multi-viewpoint%20Engineering/Model%20Validation%20and%20Analysis-1.0.0.docx

5.4 Integration and Interoperability

Transposer, Composer and Accuracy are integrated in the CTK (Core Technology Kit) of Kitalpha. Cf. Section 2.

6 Model co-evolution brick

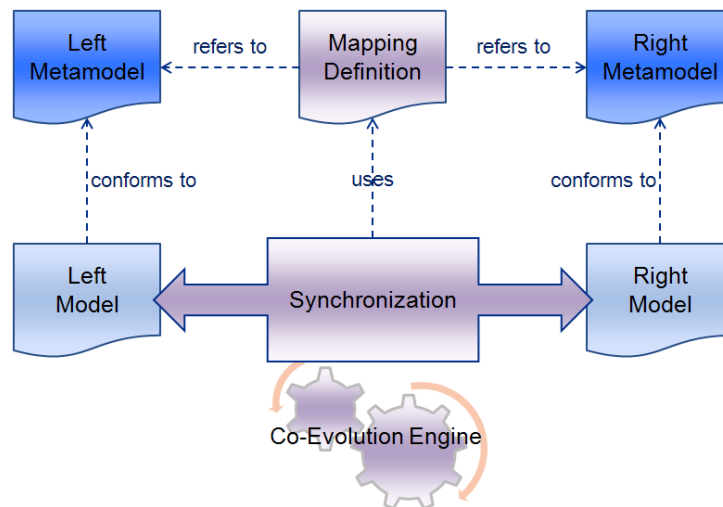
6.1 Description

Name:	Olivier Constant
Contact:	olivier.constant@thalesgroup.com - +33 1 70 28 24 82
Dependencies	This brick provides the ability to define bridges between model-based tools by defining a model-to-model transformation and executing it, when relevant, in an iterative, bidirectional and/or interactive way.
License	Eclipse Public License when mature enough, proprietary in the meantime
Additional information	To be made available as part of the EMF Diff/Merge Eclipse project (http://wiki.eclipse.org/EMF_DiffMerge)

Applying Model-Based Engineering in industrial processes often results in the production of a set of models which are technically disjoint but conceptually connected and evolve in parallel. This situation typically happens:

- when the workflow involves several modelling tools which are dedicated to different tasks, for example a system development modelling tool and tools for model-based analysis or simulation;
- when the system to design is complex enough to require the construction of different models dedicated to different sub-systems or levels of abstraction;
- when organisational boundaries between system integrators and sub-contractors raise confidentiality constraints, so each company has to work on its own private model and share only the subset related to integration interfaces;
- when a metamodel integrates many different concerns: reuse, traceability, suitability to different modelling methodologies, etc., and an abstraction of the models focussed on a single concern is required at some point for concern-specific analysis.

This brick is a technology that contributes to solve such pitfalls by supporting the parallel evolution of models, or “co-evolution”. Concretely, it allows software engineers to specify a mapping between metamodels and provides an engine that executes the mapping in order to synchronize models.



Execution principle for the model co-evolution technology

Compared to existing state-of-the-art technologies, the brick combines several capabilities which are all needed for realistic model co-evolution.

- it is based on a mapping from source metamodel(s) to target metamodel(s) defined at a high level of abstraction in order to be easy to write and maintain, notably abstracting away from the order of execution steps;
- it is expressive enough to cover cases of an arbitrary complexity;
- it supports model update so that models that have evolved synchronise instead of being fully re-generated;
- it supports semi-automatic, interactive synchronisation in the cases when human intervention cannot be avoided;
- it is also intended to support, when relevant and as much as possible, bidirectionality: the same (subset of) mapping can be executed in both directions, “source to target” or “target to source”.

6.1.1 Manual

The co-evolution technology is a new technology developed from scratch. At this stage, in accordance with the plan, only a first prototype has been developed. It has no particular license and is thus not yet available for download. It is essentially a programmatic framework whose APIs are naturally likely to evolve, so it is too early to propose a manual.

6.2 Use Case coverage and application

This brick will be used in the context of the use case 2.7 (Thales Alenia Space future satellite platform) with a focus on synchronising different engineering environments for the definition of the new avionics platform.

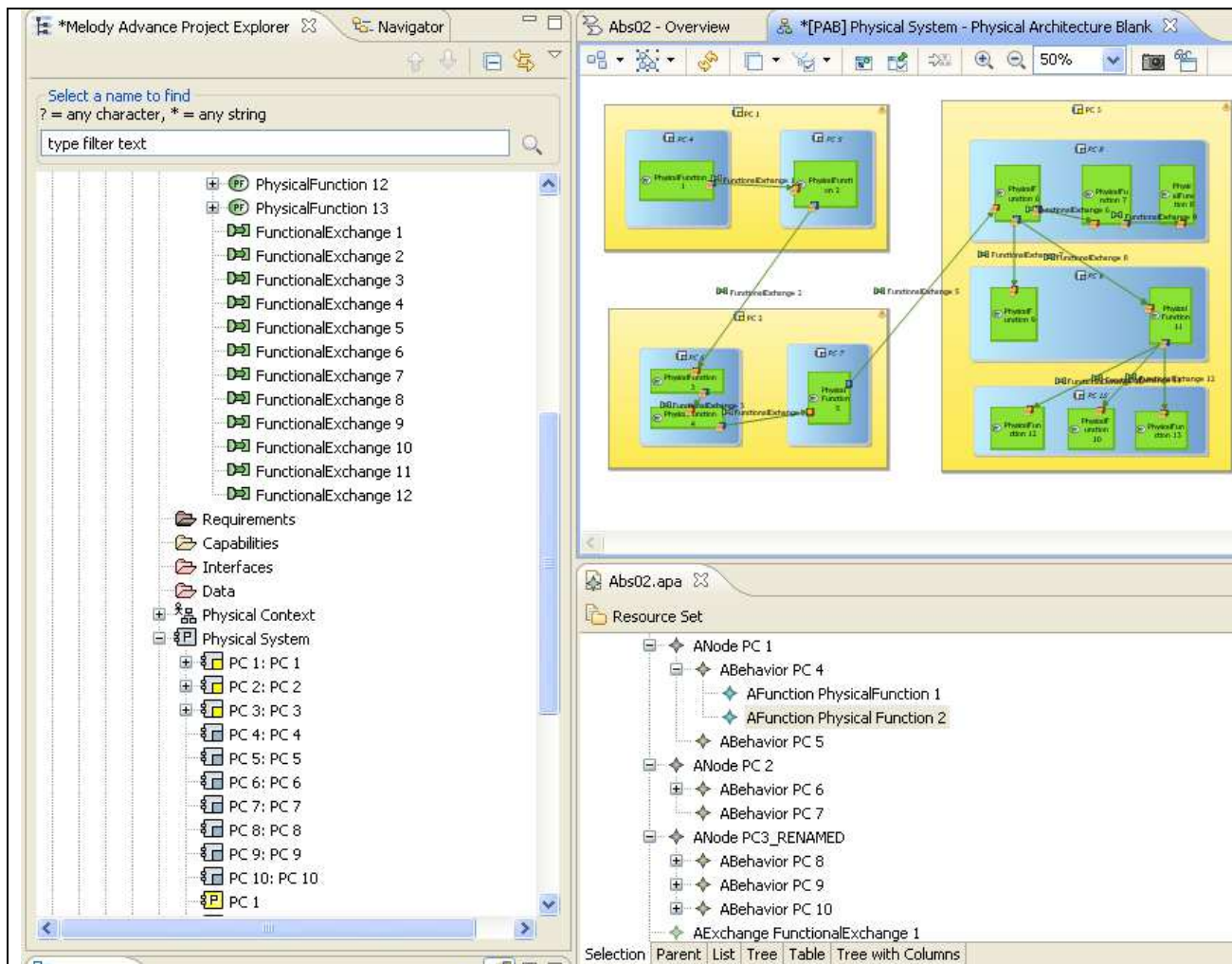
Further usage of this brick by other Crystal Use Cases will have to be identified.

6.3 General Improvement

TI NAME: Model co-evolution – State of the Art					
TI_ID	TI_0047	Kind of TI	G	Contact email	jerome.lenoir@thalesgroup.com
Description: The purpose of this TI is to provide a State of the Art about Model co-evolution. Considering the variety of existing viewpoints, how to ensure the global consistency – and, more generally, communication – between the different views of the designed system? When each of the views is being mapped to a model, this issue requires at least synchronizing heterogeneous models. The State of the Art is provided as an Annex to this document.					
Link to internal working documents:					

TI NAME: Co-Evolution Specification					
TI_ID	TI_0050	Kind of TI	G	Contact email	olivier.constant@thalesgroup.com
Description: The purpose of this TI is to provide a high-level specification for the Co-Evolution technology. This specification is available as an Annex to this document.					
Link to internal working documents:					

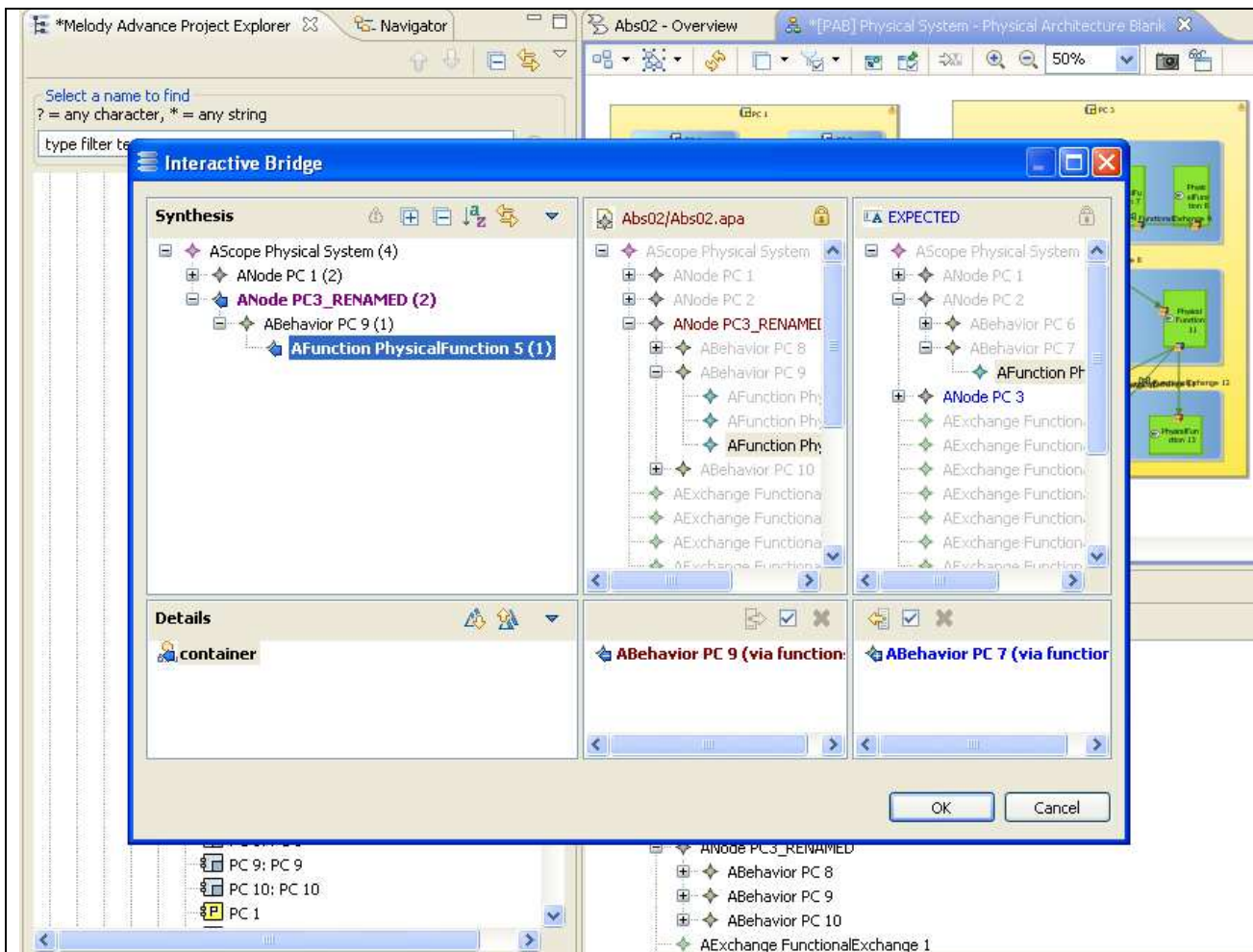
TI NAME: Co-Evolution Prototype V1					
TI_ID	CRYSTAL_TI_0049	Kind of TI	T	Contact email	olivier.constant@thalesgroup.com
Description: The co-evolution technology is a new technology developed from scratch. This TI corresponds to the first iteration of the development of the prototype. This iteration covers a programmatic Java framework for specifying mappings based on metamodels, an execution engine that supports iterativity and interactivity, and a default GUI for carrying out interactivity. It is fully functional as-is although it requires programming skills for being used. The example below demonstrates the usage of the prototype. A system model is shown in the left-hand side of the snapshot and is graphically represented in a diagram in the top right-hand corner. It is composed of components of different types (blue and yellow boxes), functions (green boxes) with ports and exchanges (arrows) between them. It involves a lot of technical elements. In the bottom right-hand corner, an editor shows an equivalent model at a higher level of abstraction thanks to a simpler metamodel that excludes technical elements.					



Example: generation of a target model (bottom right-hand corner)

This second model has been automatically generated from the first one thanks to the prototype, via a specific menu item and a mapping which has been defined between the two metamodels.

If changes are made into the first and/or the second model and the prototype is executed again, then a merge dialog opens as shown below. This dialog shows all the differences between the second model as it is and the second model as it should be according to the first model and the co-evolution mapping. The dialog allows the user to reconcile the two “second models” by deciding which changes resulting from the first model are reported to the second model and which changes made directly in the second model are preserved.



Example: iterative and interactive re-generation (synchronization)

When the OK button is pressed, the second model is saved. This synchronization mechanism can be run as many times as needed.

In order to carry out this example, the co-evolution prototype framework was used to define a *bridge* (mapping) between the two metamodels, where the bridge is composed of a set of *queries* and a set of co-evolution *rules*. The snapshot below shows an excerpt of the code. The bridge is applicable to a *PhysicalArchitecture*, which is a concept from the first metamodel. Queries are used to gather relevant source elements from the *PhysicalArchitecture*. The "Nodes query", for example, retrieves the components which are represented as yellow boxes in the diagram. Queries can be plugged onto another query, hence forming a forest-structured data flow where the data corresponds to source model elements.

```

// Bridge
final InteractiveBridge<PhysicalArchitecture, IEditableModelScope> result =
    new InteractiveBridge<PhysicalArchitecture, IEditableModelScope>(
        new MelodyDiffPolicy(), new MelodyMergePolicy(), null);
//***** QUERIES *****
// Main component query
final Query<PhysicalArchitecture, PhysicalComponent> mainPCQuery =
    new Query<PhysicalArchitecture, PhysicalComponent>() {
        public Iterator<PhysicalComponent> evaluate(
            PhysicalArchitecture source_p, IQueryEnvironment environment_p) {
            return getIterator(source_p.getOwnedPhysicalComponent());
        }
    };
result.accept(mainPCQuery);
// Nodes query
final IQuery<PhysicalComponent, Part> nodesQuery =
    new Query<PhysicalComponent, Part>() {
        public Iterator<Part> evaluate(
            PhysicalComponent source_p, IQueryEnvironment environment_p) {
            Collection<Part> result = new LinkedList<Part>();
            for (Partition partition : source_p.getOwnedPartitions()) {
                Type type = partition.getType();
                if (type instanceof PhysicalComponent &&
                    ((PhysicalComponent)type).getNature() ==
                    PhysicalComponentNature.NODE)
                    result.add((Part)partition);
            }
            return result.iterator();
        }
    };
mainPCQuery.accept(nodesQuery);

```

Programmatic definition of a mapping ("bridge") and queries

Rules, as shown in the snapshot below, aim at *describing* the target model as it should be according to the source model. They are written in an imperative style in this example, using standard Java model APIs; however, they are declarative in essence since they do not have access to any intermediate state in the construction of the target model other than in their own scope. They have access to other target elements solely via a (source, target) element mapping which is always complete, thus abstracting away from the execution order of rules. This principle is similar to declarative ATL (cf. State-of-the-Art). Rules are plugged onto queries: they are the leaves of the data flow trees.

```

//***** RULES *****/
// Rule: main PhysicalComponent -> AScope
final IRule<PhysicalComponent, AScope> mainRule =
    new Rule<PhysicalComponent, AScope>("PC2AScope") {
        public void defineTarget(PhysicalComponent source_p,
            AScope target_p, IQueryEnvironment queryEnv_p,
            IRuleEnvironment ruleEnv_p) {
            target_p.setName(source_p.getName());
        }
        public AScope createTarget() {
            return ApaFactory.eINSTANCE.createAScope();
        }
    };
mainPCQuery.accept(mainRule);
// Rule: Node Part -> ANode
final IRule<Part, ANode> nodeRule = new Rule<Part, ANode>("Part2ANode") {
    public void defineTarget(Part source_p,
        ANode target_p, IQueryEnvironment queryEnv_p,
        IRuleEnvironment ruleEnv_p) {
        // Name
        target_p.setName(source_p.getName());
        // Container
        AScope container = ruleEnv_p.get(
            (PhysicalComponent)source_p.eContainer(), mainRule);
        target_p.setOwningScope(container);
    }
    public ANode createTarget() {
        return ApaFactory.eINSTANCE.createANode();
    }
};
nodesQuery.accept(nodeRule);

```

Programmatic description of a target model via rules

The table below synthesizes the current coverage, by the prototype, of the requirements defined in Technical Item CRYSTAL_TI_0050: Co-Evolution Specification. Some requirements are covered thanks to the integration of an open-source Eclipse technology dedicated to model merging and originally developed by Thales, EMF Diff/Merge (http://wiki.eclipse.org/EMF_DiffMerge).

Status values: **OK = covered**, **OW = ongoing work**, **KO = not covered yet**.

Ref	Title	Status	Comment
1	Transformation	OK	Demonstrated in example.
2	Iterativity	OK	Demonstrated in example.
3	Interactivity	OW	Demonstrated in example, but GUI needs to mature.
4	Bidirectionality	KO	Not done yet.
5	Declarative nature	OK	Explained above.
6	No intermediate state	OK	Explained above.
7	No order	OK	Explained above.
8	Accessibility	OW	Usage feedback is needed.
9	Determinism	OW	To be confirmed.
10	Iterative vs. non-iterative	OK	Mechanism is based on EMF Diff/Merge technology.
11	Configuration of iterativity	OW	It may be possible to raise the level of abstraction even more.
12	Consistency of iterativity	OK	Mechanism is based on EMF Diff/Merge technology.
13	Data manipulation	OK	Because the prototype is a Java framework.

14	<i>Mapping expressiveness</i>	OW	Usage feedback is needed.
15	<i>Flexibility of iterativity</i>	OK	Mechanism is based on EMF Diff/Merge technology.
16	<i>Scalability</i>	OW	Overhead is limited by the fact that the prototype is a Java framework, but the raise in abstraction comes at the price of memory consumption.
17	<i>Integration</i>	OK	Because the prototype is a Java framework.
18	<i>Development environment</i>	OK	Because the prototype is a Java framework.
19	<i>Modularity and reuse</i>	OW	The fact that the prototype is a Java framework helps, but maturation is needed.
20	<i>Mapping as model</i>	KO	Not done yet.
21	<i>Metamodels</i>	OK	Because the prototype is a Java framework.
22	<i>Persistence</i>	OK	Mechanism is based on EMF Diff/Merge technology.
23	<i>Data scopes</i>	OK	Mechanism is based on EMF Diff/Merge technology.
24	<i>Trace system</i>	OW	A basic trace mechanism is provided.
25	<i>Query technologies</i>	KO	Not done yet.
26	<i>Extendible GUI</i>	KO	Not done yet.
27	<i>Parameterization</i>	OW	The fact that the prototype is a Java framework helps, but maturation is needed.
28	<i>Data/model technology adherence</i>	OW	Concern has been taken into consideration and will still be relevant in next iterations.

The coverage of requirements is expected to get improved at each iteration of the prototype. Some requirements, such as #4 on bidirectionality, are likely to be refined into several finer-grained requirements in next iterations.

In parallel to the development of the prototype, experiments for solving actual operational needs have been made and will be made again. This will ensure that usage feedback is collected in order to make progress on requirements marked as OW.

It is worth noting that a fair number of requirements such as #13, #16, #17, #18, #19 are much more easily covered if the prototype is a Java framework, which is the reason why it was developed as such. Nevertheless, other requirements such as #8, #20, #25 tend to orient the solution towards for a Domain-Specific Language (DSL). Whether a DSL should be defined on top of the current framework will thus need to be studied in next iterations.

Link to internal working documents:

6.4 Integration and Interoperability

The co-evolution technology is a solution for fine-grained interoperability between two model-based tools. Although it does not aim at providing interoperability at the scale of an engineering environment (e.g. OSLC), model coevolution may be used in the frame of interoperability in order to transform OSLC exposed models (which may be slightly different than the tool internal model but have to be kept in synchronization) or also to implement in the tool internal model modification (using REST verbs) requested at the OSLC expressivity level.

This brick itself has to integrate into and interoperate with the model transformation technology presented in Section 5. This is natural in the sense that compared to this model transformation technology, the co-evolution technology operates at a higher level of abstraction:

- its notion of declarative mapping abstracts away from the execution order of transformation rules;
- it merges the notions of model creation and model update via the notion of model description, which can be used both for creation and for update thanks to an explicit support for model diff/merge;
- it provides an explicit support for traces between arbitrary structured sets of model elements, thus extending traceability to arbitrarily complex mappings.

Conceptually, a co-evolution mapping can be integrated within the transformation technology as a transformation rule which is subject to the same scheduling policy as the other rules. Coupling the two technologies this way can be particularly relevant for large, complex transformations: it allows the transformation programmer to isolate a subset of the transformation that is suitable for being defined at a high level of abstraction with co-evolution concepts, and complete the more complex part of the transformation, which requires a lower level of abstraction, with classical transformation rules.

Note that physically, the co-evolution prototype classically consists in a set of plug-ins for the Eclipse Modelling Platform just like the model transformation technology. The technical integration of the two technologies has not been concretely done yet.

7 Integration of requirement management for multi-viewpoint engineering brick

7.1 Description

Name:	Stéphane LACRAMPE
Contact:	Stephane.lacrampe@obeo.fr - +33 2 51 13 51 42
Technical information:	This brick aims at providing an integration of requirement management capabilities into an MBE Engineering environment. This brick leverages on existing standards and tools for requirements management and focuses on integration from a tooling perspective of the requirement concern into the modelling environment.
Dependencies	This brick is based on the Eclipse platform and depends on Eclipse components like EMF and Sirius.
License	TBD
Additional information	

The work on this brick has just started, detailed description of the brick will be provided in the next version of this document.

7.1.1 Manual

The work on this brick has just started, detail description of the brick will be provided in the next version of this document.

7.2 Use Case coverage and application

This brick will be used in the context of use case 2.7 (Thales Alenia Space future satellite platform) as described in chapter 3.2 of this document.

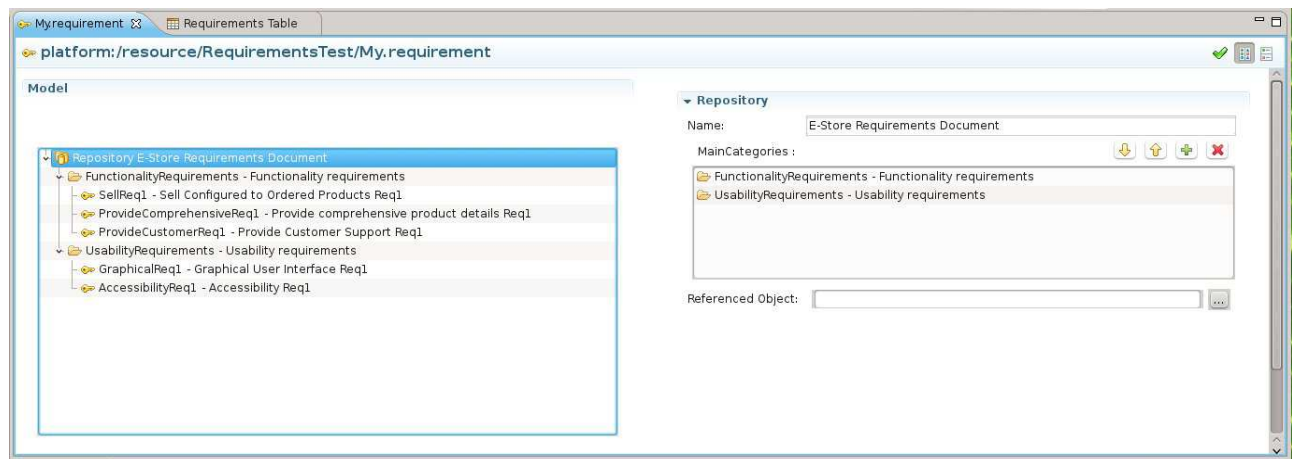
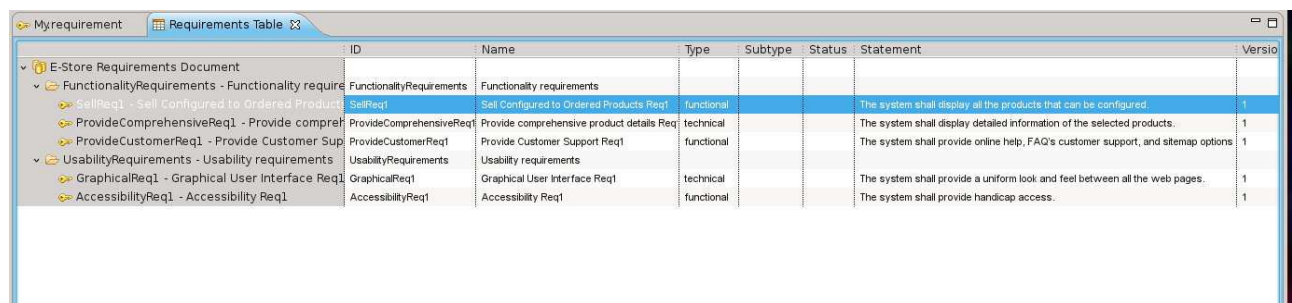
7.3 General Improvement

TI NAME: Preliminary study for requirements management integration into MBE environment					
TI_ID	TI_0077	Kind of TI	T	Contact email	Stephane.lacrampe@obeo.fr
Description: As stated earlier, this task has just started. At this stage, we have realized a first preliminary study, consisting in evaluating two internal solutions for integration of requirements management within an MBE environment :					

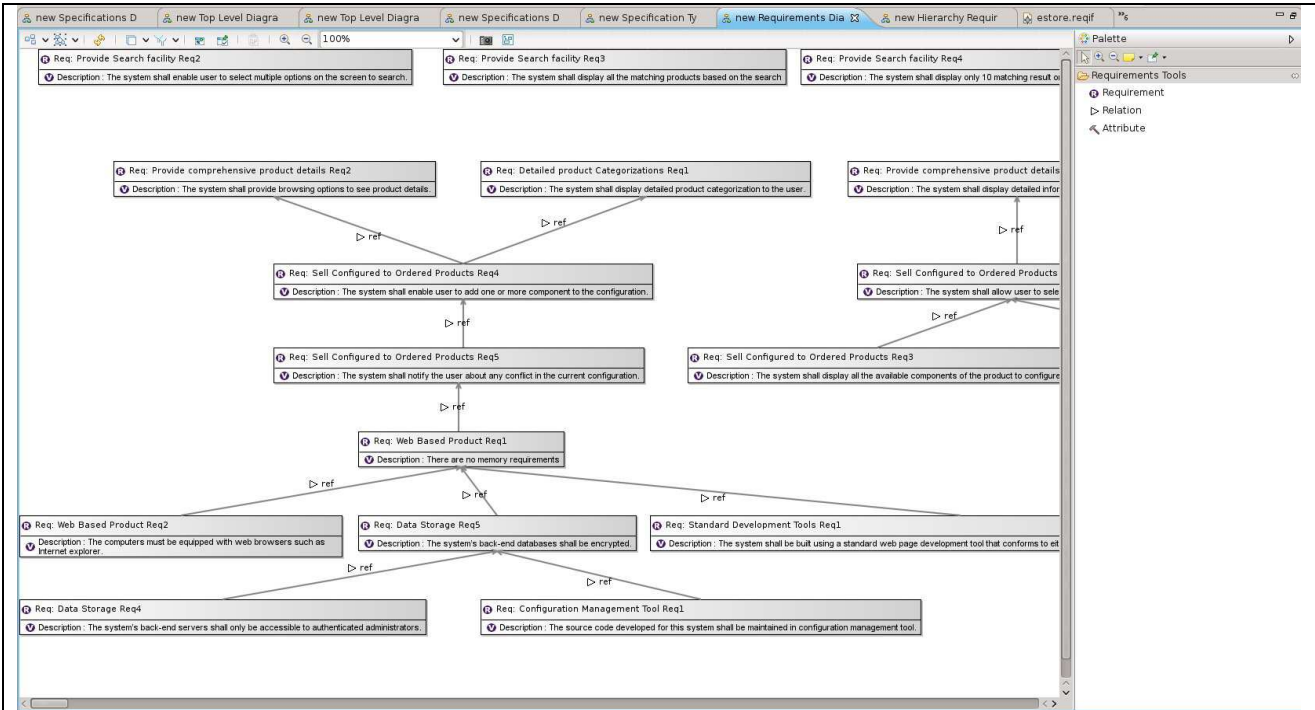
- A dedicated viewpoint based on the ReqIF format
- A dedicated viewpoint based on a simplified requirement format

Those two viewpoints have been prototyped and results have been evaluated against different concerns: representation, meta-model, tooling, genericity.

You'll find below some screenshots of the prototypes that have been realized :

ID	Name	Type	Subtype	Status	Statement	Version
FunctionalityRequirements	Functionality requirements					
SellReq1	Sell Configured to Ordered Products Req1	functional			The system shall display all the products that can be configured.	1
ProvideComprehensiveReq1	Provide comprehensive product details Req1	technical			The system shall display detailed information of the selected products.	1
ProvideCustomerReq1	Provide Customer Support Req1	functional			The system shall provide online help, FAQ's customer support, and sitemap options	1
UsabilityRequirements	Usability requirements					
GraphicalReq1	Graphical User Interface Req1	technical			The system shall provide a uniform look and feel between all the web pages.	1
AccessibilityReq1	Accessibility Req1	functional			The system shall provide handicap access.	1



	Long Name	Description	Max Length	Min	Max	Accuracy	Specified Values
✚ Datatype Definition XHTML							
✚ XHTML: XHTML datatype def	XHTML datatype def						
✚ Datatype Definition Boolean							
✚ Boolean: Boolean datatype def	Boolean datatype def						
✚ Datatype Definition Date							
✚ Date: Date datatype def	Date datatype def						
✚ Datatype Definition Enumeration							
✚ Enumeration: Enumeration datatype def	Enumeration datatype def						
✚ Datatype Definition Integer							
✚ Integer: Integer datatype def	Integer datatype def						
✚ Datatype Definition Real							
✚ Real: Real datatype def	Real datatype def			0.0	0.0		
✚ Datatype Definition String							
✚ String: T_String32k	T_String32k		32000				

Link to internal working documents:

7.4 Integration and Interoperability

Not covered at this point of the project.

8 Terms, Abbreviations and Definitions

CRYSTAL	CR itical SYST em Engineering AcceL eration
R	Report
P	Prototype
D	Demonstrator
O	Other
PU	Public
PP	Restricted to other program participants (including the JU).
RE	Restricted to a group specified by the consortium (including the JU).
CO	Confidential, only for members of the consortium (including the JU).
WP	Work Package
SP	Subproject

Table 8-1: Terms, Abbreviations and Definitions

9 References

[Czarnecki, 2009]	K. Czarnecki & J. N. Foster & Z. Hu, R. Lämmel & A. Schürr & J.F. Terwilliger; "Bidirectional transformations: A cross-discipline perspective" In <i>Theory and Practice of Model Transformations</i> . Springer Berlin Heidelberg, 2009, p. 260-283.
[Gibert, 2002]	S. Gibert & N. Lynch; "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services" In <i>ACM SIGACT News</i> , Volume 33 Issue 2; 2002, p. 51-59
[Antkiewicz, 2008]	M. Antkiewicz & K. Czarnecki; "Design Space of Heterogeneous Synchronization" In <i>Generative and Transformational Techniques in Software Engineering II</i> , Computer Science Volume 5235; p. 3-46, 2008
[Bouzitouna, 2004]	S. Bouzitouna & M.-P. Gervais; "Composition rules for PIM reuse" In <i>Proceedings on the Second European Workshop on Model Driven Architecture</i> ; 2004, p. 36-43
[Del Fabro, 2009]	M. D. Del Fabro & P. Valduriez; "Towards the efficient development of model transformations using model weaving and matching transformations" In <i>Software & Systems Modeling</i> 8, no. 3; 2009, pages 305-324.
[Czarnecki, 2003]	K. Czarnecki & S. Helsen; "Classification of model transformation approaches" In <i>Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture</i> ; 2003. P. 1-17.
[Diskin, 2014]	Z. Diskin & A. Wider & H. Gholizadeh & K. Czarnecki. "A Space of Model Synchronization Types: Symmetrization of Models Transformations and its Challenge". Technical Report, 2014.
[Di Ruscio, 2012]	D. Di Ruscio & L. Iovino & A. Pierantonio; "Coupled Evolution in Model-Driven Engineering" In <i>Software, IEEE</i> , 2012, vol. 29, no 6, p. 78-84.
[Bergmann, 2008]	G. Bergmann & A. Ökrös & I. Ráth & D. Varró; "Incremental pattern matching in the viatra model transformation system". In <i>Proceedings of the third international workshop on Graph and model transformations</i> ; pp. 25-32. ACM, 2008.
[Ökrös, 2010]	A. Ökrös & I. Ráth & D. Varró; "Synchronization of abstract and concrete syntax in domain-specific modelling languages" In <i>Software and Systems Modeling</i> , volume 9, issue 4; 2010. P. 453-471.
[Bergmann, 2008]	G. Bergmann & A. Ökrös & I. Ráth & D. Varró & G. Varró; Incremental pattern matching in the viatra model transformation system. In <i>Proceedings of the third international workshop on Graph and model transformations</i> , ACM, 2008, pp. 25-32.
[Codd, 1974]	E.F. Codd; <i>Recent investigations in relational data base systems</i> ; IFIP Congress, 1974, pp.1017-1021
[OMG, 2005]	OMG; <i>MOF2.0 query/view/transformation (QVT) adopted specification</i> ; OMG document ptc/05-11-01 (2005), available from www.omg.org
[Stevens, 2007]	P. Stevens; "Bidirectional model transformations in QVT: Semantic issues and open questions." In <i>Model Driven Engineering Languages and Systems</i> ; Springer Berlin Heidelberg, 2007. 1-15.
[Paige, 2012]	R. Paige & D. Kolovos; <i>Model transformation for fun and profit</i> , 2012
[Van Der Straeten, 2003]	R. Van Der Straeten & T. Mens & J. Simmonds & V. Jonckers; "Using description logic to maintain consistency between UML models" In «UML» 2003-The Unified Modeling Language. <i>Modeling Languages and Applications</i> ; Springer Berlin Heidelberg, 2003, pp. 326-340.
[Tsiolakis, 2000]	A. Tsiolakis & H. Ehrig; <i>Consistency analysis of UML class and sequence diagrams using attributed graph grammars</i> ; In Proc. of Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems, Berlin, 2000, p. 77-86.
[Mens, 2006]	T. Mens & R. Van Der Straeten & M. D'Hondt; "Detecting and resolving model inconsistencies using transformation dependency analysis" In <i>Model driven engineering languages and systems</i> ; Springer Berlin Heidelberg, 2006, p. 200-214.

[da Silva, 2010]	M. A. A. da Silva & A. Mougnot & X. Blanc & R. Bendraou; "Toward Automated Inconsistency Handling in Design Models" In <i>Proceedings of the 22nd international conference on Advanced information systems engineering</i> ; Springer-Verlag Berlin, 2010, Pages 348-362.
[Nissen, 1996]	H. W. Nissen & M. A. Jeusfeld & M. Jarke <i>et al.</i> ; "Managing multiple requirements perspectives with metamodels" In <i>Software</i> ; IEEE, 1996, vol. 13, no 2, p. 37-48.
[Del Fabro, 2007]	M. D. Del Fabro & P. Valduriez; "Semi-automatic model integration using matching transformations and weaving models" In <i>Proceedings of the 2007 ACM symposium on Applied computing</i> ; ACM, 2007, p. 963-970.
[Falleri, 2008]	J. R. Falleri & M. Huchard & M. Lafourcade & C. Nebut; "Metamodel matching for automatic model transformation generation" In <i>Model Driven Engineering Languages and Systems</i> ; 2008, p. 326-340.
[Varró, 2007]	D. Varró & Z. Balogh; "Automating model transformation by example using inductive logic programming" In <i>Proceedings of the 2007 ACM symposium on Applied computing</i> ; ACM, 2007, p. 978-984.
[Sun, 2011]	Y. Sun; <i>Model Transformation by Demonstration: A User-centric Approach to Support Model Evolution</i> ; (Doctoral dissertation, The University of Alabama), 2011.
[Blanc, 2008]	X. Blanc & I. Mounier & A. Mougnot & T. Mens; "Detecting model inconsistency through operation-based model construction" In <i>Software Engineering</i> ; 2008. ICSE'08. ACM/IEEE 30th International Conference on. IEEE, 2008, p. 511-520.
[Spanoudakis, 2001]	G. Spanoudakis & A. Zisman; "Inconsistency management in software engineering: Survey and open research issues" In <i>Handbook of software engineering and knowledge engineering</i> ; 2001, vol. 1, p. 329-380.
[Guerra, 2011]	E. Guerra & J. de Lara & D. S. Kolovos <i>et al.</i> ; "Engineering model transformations with transML" In <i>Software & Systems Modeling</i> ; 2011, p. 1-23.
[ISO-42010, 2007]	ISO. ISO/IEC 42010 <i>Systems and Software Engineering — Architectural Description</i> , July 2007.
[Rundensteiner, 2000]	E. A. Rundensteiner & A. Koeller & X. Zhang; Maintaining data warehouses over changing information sources. <i>Communications of the ACM</i> , 2000, vol. 43, no 6, p. 57-62.
[Rizzi, 2006]	S. Rizzi & A. Abelló & J. Lechtenbörger & J. Trujillo; Research in data warehouse modeling and design: dead or alive?. In <i>Proceedings of the 9th ACM international workshop on Data warehousing and OLAP</i> , ACM, 2006, pp. 3-10.
[Morzy, 2004]	T. Morzy & R. Wrembel; On querying versions of multiversion data warehouse. In <i>Proceedings of the 7th ACM international workshop on Data warehousing and OLAP</i> . ACM, 2004, pp. 92-101.
[Wache, 2001]	H. Wache <i>et al.</i> ; "Ontology-based integration of information-a survey of existing approaches". In <i>IJCAI-01 workshop: ontologies and information sharing</i> ; 2001, pp. 108-117.
[Chen, 2011]	H. Chen & H. Liao ; "A Survey to View Update Problem". In <i>International Journal of Computer Theory and Engineering</i> ; Vol.3, No.1, February, 2011
[Keller, 1986]	A.M. Keller; The role of semantics in translating view updates in <i>Computer</i> ; 19(1), 63-73; IEEE. 1986.
[Sheth, 1988]	P. Sheth & J. A. Larson & E. Watkins; "TAILOR, a tool for updating views". In <i>Advances in Database Technology—EDBT'88</i> . Springer Berlin Heidelberg, 1988. 190-213.
[Shu, 2000]	H. Shu; "Using constraint satisfaction for view update". In <i>Journal of Intelligent Information Systems</i> 15.2; 2000, p. 147-173.
[Ellis, 1989]	C. A. Ellis & S. J. Gibbs; "Concurrency Control in Groupware Systems". In <i>ACM SIGMOD Record</i> , 18(2); 1989, p. 399-407.
[Mah, 2010]	A. Mah & S. Lassen; "Google Wave Operational Transformation"; 2010.
[Fraser, 2009]	N. Fraser; "Differential Synchronization", 2009.

10 Annex

10.1 Annex I: State of the Art of Model Synchronization

Model transformation is a mature topic in MDE, which consists in defining a set of operations on describing formal relationships between inputs and outputs, using them to produce a model (or possibly data, code, etc.) from an input model, typically offline. Model transformation techniques involve a source model (or set of models) and destination models (or set of models). These sets can be overlapping, in case of endogenous transformation, but the notion of causality is widely accepted: the source exists before the transformation and the destination (in its final state) exists after the transformation. A typical example of these techniques is code generation, illustrated by Figure 10-1.

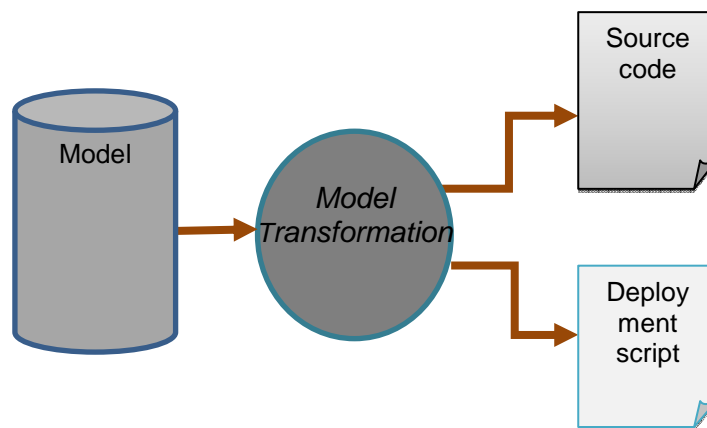


Figure 10-1: Model transformation for code generation

As opposed to model transformation, model synchronization does not refer to a particular technical solution, but to a kind of problem in which model transformation could be a part of the solution. Typically, problems requiring model synchronization involves *semantically overlapping* models that *exist in parallel* and *evolve concurrently* [Czarnecki, 2009]. An example of model synchronization needs is data warehouse, where multiple databases distributed on different servers are maintaining different schemas consistent according to a schema mapping defined by a database expert, as illustrated in

Figure 10-2, extracted from [Rundensteiner, 2000].

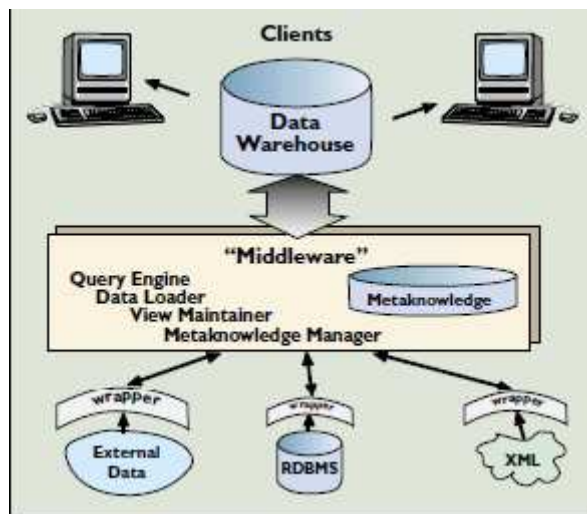


Figure 10-2: Data Warehouse

Since the models involved in model synchronization typically conform to different metamodels (since they model different concerns or scopes of the system), the nature of the impact of a transformation proceeds from classical model transformation techniques, although it would be quite easier to apply transformation on model updates than on the whole model.

Another more original aspect of model synchronization is that each model is of equal importance – which means there is no more causality paradigm between the different models – and modification of any model must be impacted on the other ones. This raises the necessity of incrementality – that is, non-destructive updates of so-called target models.

Update Mapping

Although model generation can be done by using the same technologies than model synchronization, a more generic case of the latter issue involves a set of pre-existing models. For each possible update on each of the model, corresponding updates must be defined for a subset of the other synchronized models. In the worst case, all other models must be updated, but in practical cases knowledge from pre-existing process can be used in order to reduce the number of actual dependencies.

In a restrictive acceptance, possible updates are defined by the set of editing tools and operations existing on the model. In this point-of-view, *operation-based* definition of a corresponding mapping model would be an efficient solution, since it allows reducing the overall number of rules and make them quite easier to write and maintain than the model transformation-based solutions.

However, one is likely to wish some import options in these tools, thus reducing the problem to classical model transformation – at least concerning the rule definition. Computation of a delta between two models in terms of existing operations is likely to be an answer to the latter issue.

Incrementality

As opposed to classical model transformation, model synchronization involves co-existing models with semantical overlapping. Thus, edition of a model must induce a change on at least one of the other synchronized models. This change, however, must preserve the existing model, rather than regenerate it, and have only minimum impact on the model's objects for efficiency sake. This latter concern also raises the issue of scoping.

Version	Nature	Date	Page
V01.00	R	2014-04-2830	62 of 117

Concurrency

Although it is not the case in current implementations due to performance limitations, an important application of heterogeneous model synchronization is concurrent edition and distribution amongst different servers. It raises issues of consistency, availability and fault detection. Those problems have been studied extensively in both academics and industry, with two notable results: all properties cannot be ensured in the same time, and compromises can be found with any two of these properties (CAP theorem and developments, as explained in [Gibert, 2002]). These compromises, however, are highly application and architecture-dependent.

Which compromise can be acceptable in the context of distributed models (and their respective tools like editors or analyzers) is a subject who has been studied in the database fields with distributed databases, and later data warehouse, where the original aim was to increase query performance on unstable networks. In order to achieve this result, techniques coming from distributed relational databases world are generally used [Rizzi, 2006]. We claim that these results can be applied efficiently to the field of MDE.

Positioning heterogeneous model synchronization

Heterogeneous model synchronization variants can be form under many labels. One of the most frequent is *model federation*. This last term describes the same technical challenges at modelling level (aligning heterogeneous models), but differs in its lack of consideration for the concurrency and update issues. *Bidirectional transformation* refers to the same scope, yet focus on the technical aspect instead of the high-level objective. Another near subject is the *collaborative working*, which focuses more on concurrency and incrementality, but is generally not done on heterogeneous models.

The field of *model coevolution* is often associated to model synchronization. Although it can be seen as a much degenerated case of model synchronization (in with a model and its metamodel are the models synchronized), it relies on a-priori knowledge of the relation between the models (the metamodel conformance relationship) which cannot exist in more generic cases. Thus techniques from model coevolution are not applicable in model synchronization.

Distributed Relational Databases techniques used for Data warehouse synchronization needs share similarities with heterogeneous model synchronization, since it can address models following different formalism, raise important issues of concurrency and distribution, and have sources likely to evolve over the data warehouse lifecycle [Morzy, 2004]. A major difference however is that the data from the different sources have little semantical overlapping, and that data warehouse techniques would limit themselves to consistency checking rather than propagation of modification. Hence no complex mappings are necessary, as opposed to heterogeneous model synchronization. This issue is addressed by ontologies which are, in many respects, continuators of relational algebra-based approaches for the data warehouse problem [Wache, 2001]. Furthermore, data warehouses techniques generally aim to optimize non-modification queries (i.e. SQL *select*) rather than updates.

An important aspect of the model synchronization is that it is motivated by a process-based approach, as opposed to model federation that is more technically-oriented. Model synchronization is closely tied to a multi-viewpoint, multi-metamodel approach of modelling, since unique-metamodel approaches are inefficient in capturing the whole operational issue and solution system in practical cases.

Antkiewicz and Czarnecki proposed in [Antkiewicz, 2008] a taxonomy of heterogeneous model synchronization techniques. Although it covers only a subset of the scope of actual heterogeneous model synchronization (including currency issue, consistency checking and correspondences specification), it is a complete review of the state of the art in terms of synchronization mechanisms. The different proposed synchronizers (mechanisms of model synchronization) range from unidirectional one-to-one non-incremental synchronizer to bidirectional, source-incremental, many-to-many update-based synchronizer with reconciliation. Trade-offs of the different techniques are discussed in their article.

Version	Nature	Date	Page
V01.00	R	2014-04-2830	63 of 117

Finally, in a recent and important article, [Diskin, 2014] defined a taxonomy of the *objectives* of model synchronization, according to the kind of desired synchronization. In this article, the authors describe three dimensions of model synchronization:

- *Informational symmetry*, which describes the correspondence relationship between elements of source and target models (using a database and its view as an example of *informational domination*);
- *Organizational symmetry*, which describes the authority of a model over another one (describing various possible organizational domination kinds between a commercial model and the corresponding technical model);
- *Incrementality*.

The notion of organizational or informational symmetry developed in the article is important since it opens ways to capture both semantical links between the models (informational symmetry) as well as process knowledge (organizational symmetry). The latter part is an important aspect of synchronization engineering which should be researched further in order to offer a sound framework for developing synchronization usable in the long run.

To these dimensions, we claim that the *concurrency* dimension should also be added.

In the following sections, we present the two former aspects of model synchronization map to current knowledge in industrial and academic fields.

10.1.1 Organizational aspects

10.1.1.1 One system – multiple concerns and stakeholders

In the industry complex system design involves many stakeholders (software and system architects, domain experts, managers, etc. [Rozanski, 2013]). These stakeholders have specific concerns (i.e. security, evolution, performance, cost efficiency...), that are not necessarily shared. Most of the time a stakeholder has a domain-shaped perspective on the system which makes difficult to communicate with other actors.

These stakeholders work on different stages of the analysis and design process, from requirement definition to implementation. During these stages, a large volume of partial descriptions of the system are produced in varied formalisms (formalized text document like System Requirement Specification for instance, spreadsheet, pictures, binary code, etc.). These partial descriptions corresponding to actors concerns are often named “views” in literature. In practical case, they are also associated with existing tools, which assist the actors in their design or analysis task. Views are descriptions of the system conforming to a viewpoint.

This raise an issue: considering the variety of the existing viewpoints, how to ensure the global consistency - and, more generally, communication - between the different views of the designed system? When each of the views is being mapped to a model, this issue requires at least synchronizing heterogeneous models. This issue proved to be serious, and is the cause of many industry failures.

There are multiple approaches on how a multi-viewpoint framework should organize its data. The two extrema of these strategies are:

- In one hand the use of a single, unified model which contains the union of all the concepts needed by the views,
- and in the other hand the use of specific models dedicated to each view.

Apart from those approaches, a common trade-off is to use a limited set of models, each one being used by multiples views. Although it reduces the effort, it does not address the issue of consistency between the models behind the different views.

10.1.1.2 View Correspondences and standard

The standard ISO-IEC-IEEE 42010 [ISO-42010, 2007] describes the notion of *architecture description*, which metamodel is illustrated by Figure 10-3. An architecture description is, according to this standard, a composition of *views* and *correspondence rules*. A correspondence rule governs a set of *correspondences* relating two or more *architecture elements*. Typically, a correspondence defines a semantic relation between elements from different views. In the latter sections of the present document, it will always be used in this meaning.

As can be seen in the figure, a multi-model approach is implied by the standard – without hypothesis on the homogeneity of the models. Hence any of the previously cited approaches for ensuring consistency of the different viewpoints can be used compliantly to ISO-IEC-IEEE 42010.

Version	Nature	Date	Page
V01.00	R	2014-04-2830	65 of 117

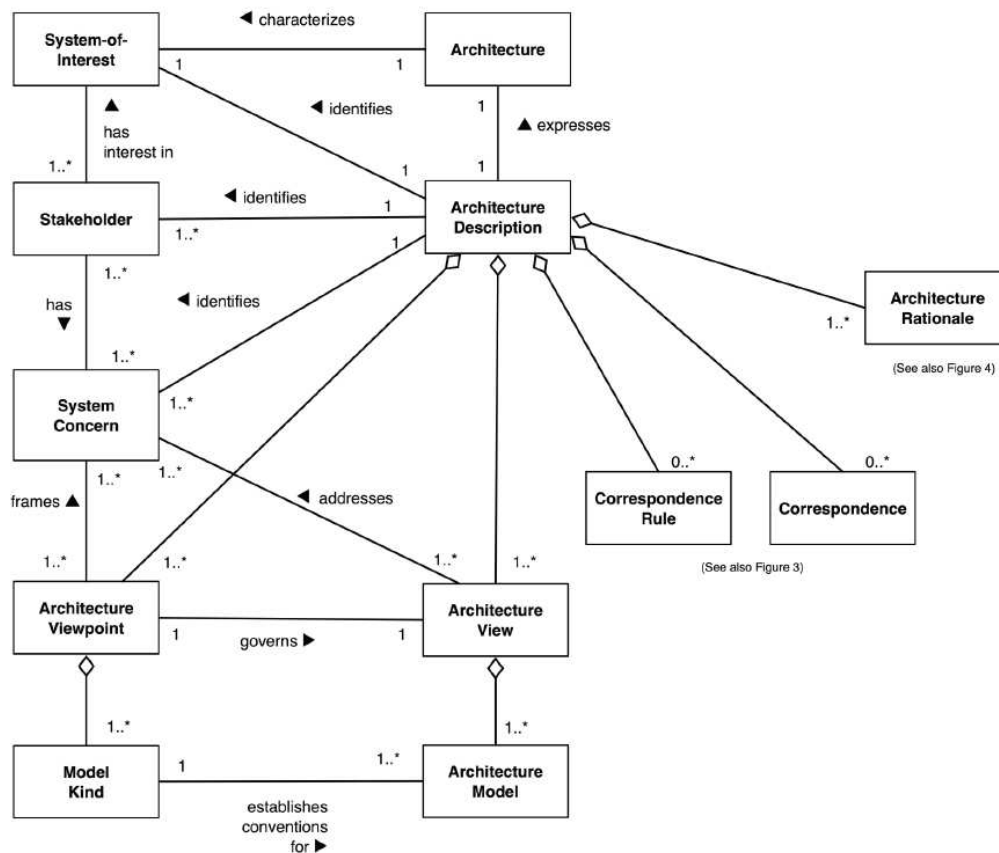


Figure 10-3: Architecture metamodel - ISO 42010

The following sections will briefly describe the main strategies in order to guaranty constancy between the different views and (hence) models.

10.1.1.3 Model coevolution and Heterogeneous model synchronization

Model-driven engineering does not only consists in modelling formally abstract views of the system, but also in maintaining the information contained in the associated models over the product lifecycle and possibly even over a longer period. However, over a long period, the metamodels which these models are conforming to are likely to evolve, for instance to allow the addition of non-previously available features on the product.

This raises the issue of how to conform existing models to evolving metamodels. *Model coevolution* is the term commonly used to describe the techniques addressing this issue. It is quite different from *heterogeneous model synchronization* for two reasons:

- It can rely on knowledge of the semantic of the relation between the two models (the *conformance* relation between a model and its metamodel) ;
- It is an offline, non-dynamic activity, and thus does not concern itself much with performance and concurrency issues.

The former point allows to infer operations (as *creation*, *deletion* or *updated*) on model from the metamodel modification. For instance, the deletion of a meta-element will generally trigger the deletion of all associated elements (although inheritance can make the problem more complex).

Heterogeneous model synchronization, in the other hand, aims to “synchronize” models whose relations are not known *a priori*. More importantly, the relations’ actual semantics are not known either, which means that that correspondence specification must be specified manually. Furthermore, model synchronization is done “online” (i.e. at model edition time) and raises issues of distribution and concurrency. Heterogeneous model transformation is further explained in the next chapters.

Contrary to what its official title suggests, the actual scope of this document is to describe heterogeneous model synchronization, not model coevolution. The latter subject was already covered by many studies, and a recent review of existing tools has been done by [Di Ruscio, 2012].

10.1.1.4 Views consistency

This section describes different families of solutions for ensuring consistency between the different views on the system and summarizes their benefits and flaws.

10.1.1.4.1 Process-based solution

A solution for the consistency problem is to ensure consistency by process. This somewhat by-default approach consists in forbidding concurrent change on the different models. The design process can still be incremental, but each task is strictly scheduled following a causal order. Rework does involve a full new iteration of the process. Consistency between the consecutive models is usually ensured either by manual work or by generating the new model from the previous one – which is a degenerated and simpler case of model coevolution. Although it is often used in existing design process and offers good reliability characteristics, it does have some drawback.

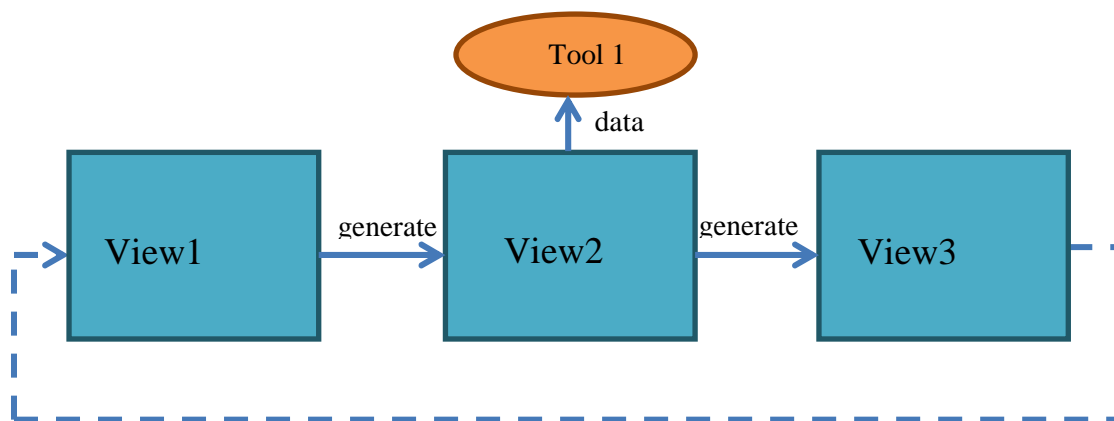


Figure 10-4: process-based approach

Figure 10-4 illustrates the process-based approach. In this example, each view accesses to a model which is used to generate (automatically or manually) the next model, accessed by a dedicated view. Each view can (and generally is) be connected to tools (for model analysis, simulation, etc.). Since the process is sequential, each modification on *view1* implies a new development cycle (*view1*, *view2* and *view3*). Note that in this approach, the viewpoints and models are generally defined by the tools, and thus are on-the-shelf components.

10.1.1.4.1.1 Very slow and rigid process

Since the process is made of a succession of protected sections, the cost of a correction is important: one must finish entirely the current task before integrating changes from the previous task, regardless of the actual progress and dependency of the current task. This approach does not model the dependencies between data and does not allow skipping any design step.

10.1.1.4.1.2 Error-prone manual traceability

Manually tracing down the links between two models is an error-prone process, which is also time-consuming. Traceability matrixes are hard to maintain in case of change of the source model.

10.1.1.4.1.3 Conserving changes from previous version

Code generation from models is a technique which is well mastered by the industry, and thus quite reliable. However, it does not natively allow keeping track of the changes in case of reengineering (i.e. any iteration of

the process after the initial one). It is possible to find workarounds, but then it involves issues as complex as coevolution.

10.1.1.4.2 Unique model solution

It was a trend in industry for solutions of kind one-model-fits-all, which postulated that viewpoints could be emulated by different layout of the same model [Gamma, 2008]. In this approach, all the data describing the system were put in a single repository, following one unique schema. Although it does address the consistency issue, this approach has major flaws.

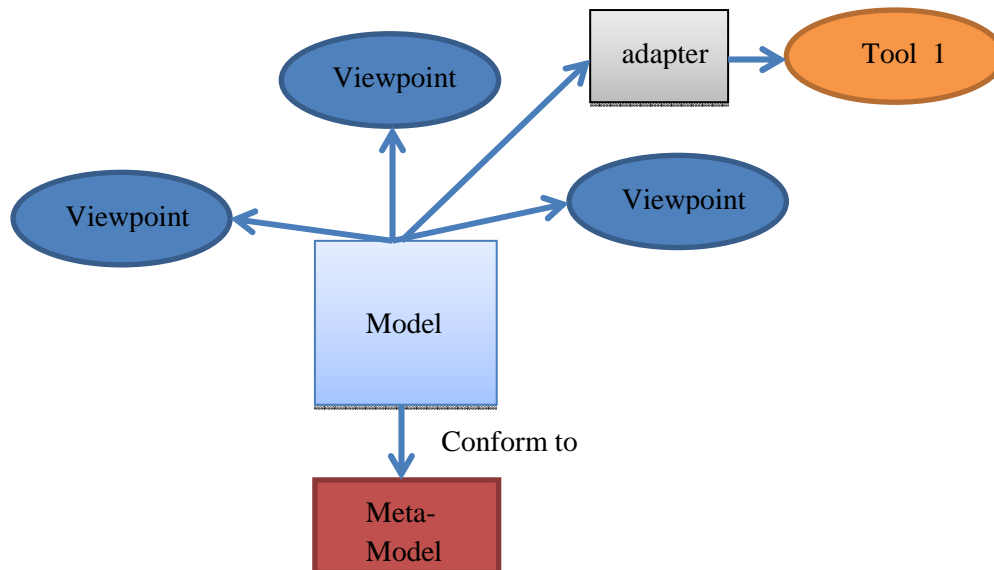


Figure 10-5: Single-metamodel approach

Figure 10-5 illustrates the single-metamodel approach. In this example, a single model is connected to different viewpoints and tools. For each tool, an adapter needs to be developed.

A more realistic variant of this solution is the unique meta-model but distributed model solution. In this case, the data share a common structure, but parts of the model can be partitioned across the different viewpoints.

10.1.1.4.2.1 Adapting legacy process

From the actor point of view, and considering an already-existing process, it does involve throwing away existing modelling tools (in the larger meaning of the term) for new ones, thus imposing over-costs and impacting negatively on designers' efficiency. Moreover, it involves a considerable work to build mapping between the unique metamodel and existing analysis tools, since they rely heavily on underlying meta-model.

10.1.1.4.2.2 Increasing communications

Each modification impacting the global model, it must be communicated to other users, even if this information is not actually useful in their own perspective. Thus, it involves many unnecessary communications, which may cause board band overload and increases the framework latency.

This aspect can be fixed by distributing specific parts of the model amongst the different users and synchronizing locally only those parts.

10.1.1.4.2.3 Inherent complexity of a global model

Planning to design a global model does underestimated the intricate distance between different perspectives, which make a common model being at the same time very difficult to produce and quite hard to build new viewpoints on. Furthermore, an over-complex model would be quite difficult to maintain.

In actual, modern systems, this approach cannot actually be used completely. The wide variety of specialties involved in the system design make it impossible for a few persons or even for a single enterprise to master all the concepts necessary for their efficient design. Thus the design of a single, universal, semantically strong and actually usable meta-model is but a myth. This tends to lead toward semantically weak models interpreted by experts – and in this case the actual knowledge is not captured in the model, but stays hidden in the expertise.

10.1.1.4.2.4 Meta-model and model Reusability

Since all the data follow a monolithic meta-model, it is unlikely than this definition can be reused in another context. The same issue happens for the model, in the single-model variant.

10.1.1.4.3 Multiple meta-models solution

The multiple meta-model solution consists in maintaining a set of different models from distinct meta-models and synchronizing them in some way. We refer to this solution as *model coevolution*. This solution allows designing concern-oriented meta-models, which are more easily exploitable by diverse tools than universal meta-models. Another interest of this solution is the possibility to integrate legacy tools, modellers and even models into the system representation, which greatly increase the global reusability of the system model and make the process transparent for end-users (i.e. experts, architects, etc.).

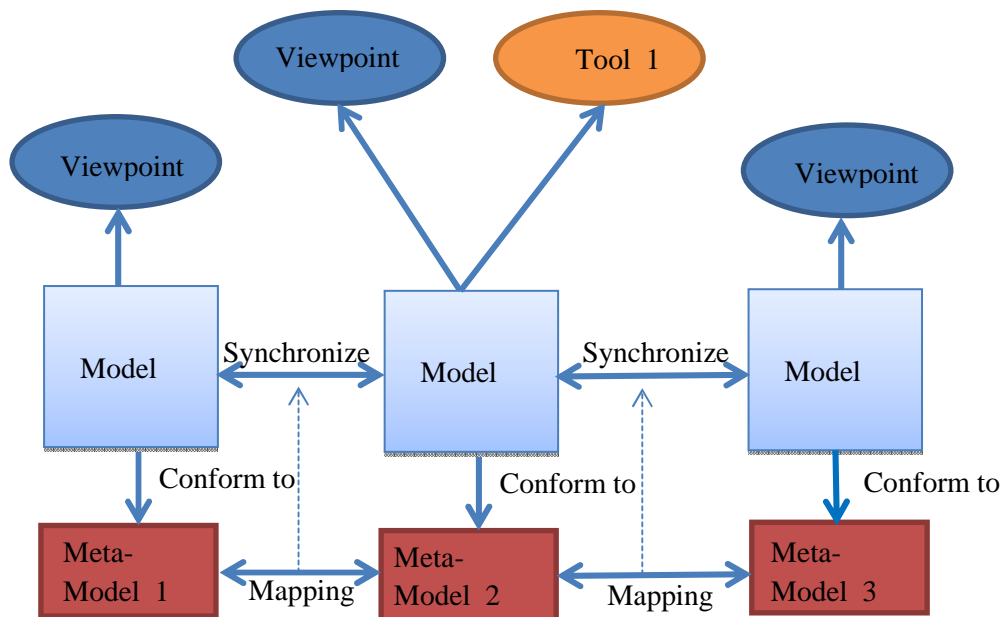


Figure 10-6: Multiple metamodels approach

Figure 10-5 illustrates this last approach. In this example, the different aspects of the system are formalized conforming three different meta-models. Since these aspects are overlapping, some elements from the conforming models must be synchronized. This synchronization can be done at model level, but is more effectively done at meta-model level.

Note that in practical cases, the coevolution solutions can cover a wide spectrum of actual implementations, from a fully-distributed pair-to-pair system to centralized solutions. This aspect is cross-cutting to the current

concern, and will be developed latter in this document. Although these solutions display interesting features and integrate well into an existing process, it does have some drawbacks.

10.1.1.4.3.1 Difficulty to map diverse models

In the same way that it is difficult to design a unique model addressing all the concepts of many concerns, it is difficult to define a mapping between two genuinely heterogeneous meta-models. This flaw can be addressed by defining only one-to-one mapping, or any trade-off between this solution and the unique mapping (metamodel –to–metamodel). This, however, raise the issue of the number of mapping of design.

10.1.1.4.3.2 Number of mappings de design

If a solution is chosen going to the one-to-one side of the metamodel mapping spectrum, the number of mappings raise up to (n-1) factorial, with n being the number of metamodels used. This flaw can be reduced by using the knowledge from process – all metamodels are not necessarily connected. From this point arises an interesting consideration: the very principle and prerequisite of coevolution approaches is the precise understanding of metamodel causality.

10.1.1.4.3.3 Communications issues

The communication issue with the coevolution approach is less severe than the one-model approach, since the precise knowledge of model dependencies (as modelled in the mapping model) allows a minimal number of communications. It also, however, does raise the issue of any distributed systems stating that between consistency, availability and partition-tolerance, only two can be guaranteed by the same system.

10.1.1.4.4 A classification of solutions

We briefly described the main characteristics of the main approaches addressing the issue of views synchronization, focusing on their flaws. We propose to use them in order to build a set of criteria describing their respective interest, as shown in Table 2-2.

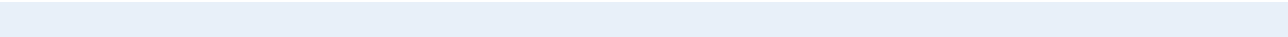
	Process-based	Single meta-model, monolithic model	Single meta-model, distributed model	Multiple-metamodel
Concurrent work enabled	No	Yes		Yes
Error-prone process	Possibly	No		No
Change conservation	Not natively	Yes		Yes
Requires existing tools adaptation	No	Yes		No
Distribution issue	No	No	Yes	Yes
Reusability	Yes	Unlikely		Yes
Meta-model design difficulty	Usual : requires domain knowledge	Very hard : requires multi-domain knowledge needed and interaction capture		Usual : requires domain knowledge
Concern mapping difficulty	Yes, distributed amongst the different stages	Yes, requires mapping between the meta-model and all the viewpoints		Yes, between the different models
Number of mappings	Process-driven : follows the data dependency	Limited: follows the number of tools and layouts.		Theoretically important, but practically process-driven : follows the data dependency



Table 2-2: classification of solutions

From the analysis of these results, we can conclude that the multi-metamodel solution shares the best aspects of both process-driven solution and single-meta-model solution. As a matter of fact, multi-model solution is a solution driven by operational concerns, while the single metamodel solution is driven by (self) technical concerns. The latter solution will require designers to bend their habits in order to follow the new, more efficient way, while the former solution will use technologies from the previous one in order to adapt to the designers ways and make them more efficient (by adding concurrency and native change conservation and by increasing traceability).

We advocate that an approach which is the less intrusive into an existing process is more likely to be adopted by the different system design actors. This is the reason why the following parts of the present document will describe more extensively the main technical requisite of multi-metamodel solution: model synchronization.



10.1.2 Establishing semantical correspondences between models

10.1.2.1 Defining correspondence rules

The ISO/IEC 42010 standard defines a *correspondence* class which links two or more architecture elements from different models in the same architecture description. These correspondences are typically used to describe a *semantical relation* between the linked elements. Although the most straight-forward kind of relation is the identity (multiple elements from different models being the same element in the actual, global system), the set of possible relations does not restrict to it.

Note that, since the ISO 42010 does not specify whether the models conform to the same meta-model (homogeneous models) or not (heterogeneous models), the identity case is not necessarily relevant in the set of possible correspondence relations, unless the models are already strongly aligned – which is not always the case in large projects. Nevertheless, in the context of heterogeneous models synchronization, it is a legit and important relation.

If the semantical overlap between models was to be limited to identities simple existence links would be sufficient to implement the correspondences, more complex relations require either dedicated languages such as transformation languages or user input. In the latter case, correspondence could be dynamic (computed) rather than static (described with links). Although we make the assumption that the correspondences are described at metamodel level (M2), it is a common practice to describe them at instance level (M1). Since the ISO/IEC 42010 does ground explicitly on MDA, this practice is legit from the standard's point-of-view. The following sections explore the issues of identity links-based solutions.

10.1.2.2 Modelling correspondence rules at meta-level

The most common problem in correspondence definition at metamodel is to describe identity between sets of elements of different metamodels. In a MOF-compliant metamodel, these elements can be of three kinds: *links*, *instances* and *properties*. This issue was explored in the context of Atlas Model Weaver (AMW¹), an open-source Eclipse plugin primary aimed at model transformation. A meta-model for correspondence named *weaving meta-model* was proposed. The weaving meta-model allows to describe weaving (correspondence) models, which link elements from the source and target meta-models.

From this weaving model, correspondences can be inferred between source and destination models with a correspondence engine. Although these correspondences are generally static, the process can also be used in a dynamic way. Figure 10-7 illustrates the different stages the weaving process in a MOF context. Note that such method can be only used in a MDA context, with an explicit meta-metamodel.

Defining the correspondences at the “meta” level (M2) is an important improvement as compared to declare them at instance level, since the number of links (or rules, in the context of a transformation language) is basically proportional to the number of elements, which ranges in an industrial context from about 200 meta-elements to about 100000 elements. More importantly, it also allows reuse of existing (meta) correspondences between different versions of the same product, or to a lesser extend between different products of the same business line. It, however, implies a new operation: the *matching*, which consists in computing the sets of all instances that are legit sources for a given transformation. There is obviously no instance matching operation on an instance-based correspondence.

¹ <http://www.eclipse.org/gmt/amw/>

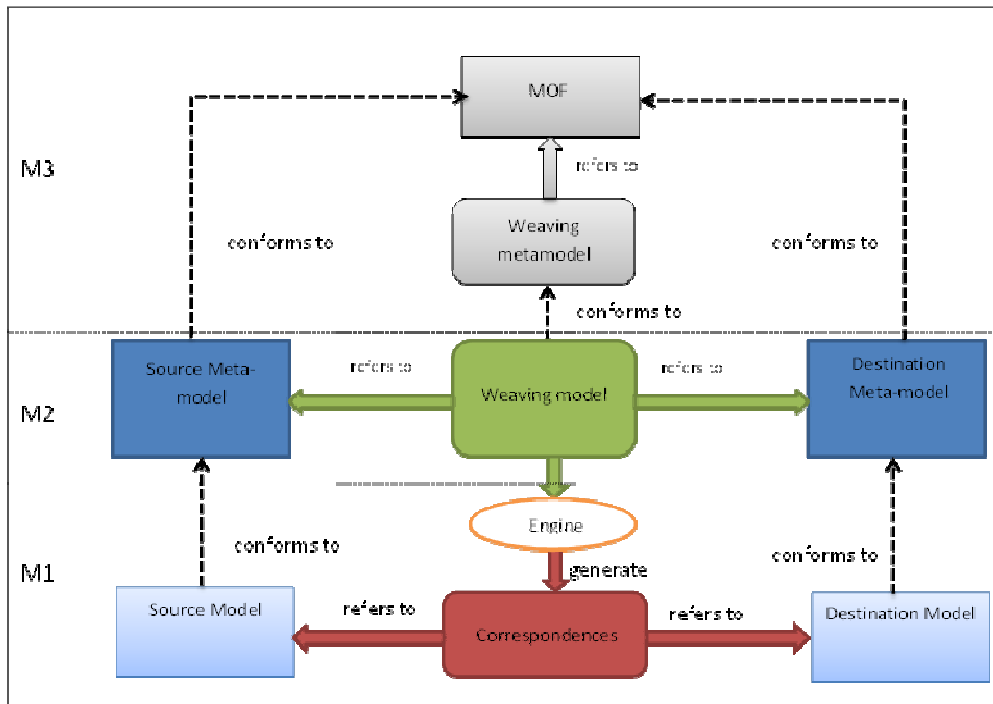


Figure 10-7: Weaving the meta-models

In academic and industry literature (as outlined in [Bouzitouna, 2004]), the whole process of establishing correspondences at meta-model level and use them to compute and/or apply correspondences at instance level is globally named composition, and is divided into three categories of rules:

- **Correspondence rules**, used to define correspondence between the related model elements ;
- **Combination rules**, used to describe *how* elements from the different models sharing correspondence links must be combined ;
- **Replacement rules**, used to define which elements in source models should be replaced by other ones.

The following section will explore the two former issues, as *replacement* is trivial.

10.1.2.2.1 Correspondence rules

10.1.2.2.1.1 Modelling aligned metamodels overlap

As outlined by [Antkiewicz, 2008], there are two approaches in establishing correspondence: *non-structural matching* and *structural matching*.

Non-structural matching is only fit to model that are really aligned, that is, whose transformation operations are bijectives and in which common instances are different view of the same responsiveness. In this case, the elements are associated by some kind of identifier. Although technically straightforward, this situation is quite common in MDE, as many viewpoints were originally designed according to a process which emphasized collaborative work. In this case, establishing a matching language is quite simple, since it does not require exploring the instances graph. Note that not all bijective situations allow a non-structural matching: it is possible that, although the elements are in bijection, they do not possess unique identifier. In this case, structural matching must be performed instead of non-structural matching. Non-structural matching rules are simpler to write and can be matched in $O(n)$. It does not require a complex exploration of an element responsiveness, but only to interrogate the elements attributes and properties. This approach, however, although fitting for creating correspondence links, presents some drawback in the context of

models evolution. For instance, the deletion and recreation of an element in a source model would probably generate a new identifier, and thus would not be matched against the previously-matched correspondence.

The case of structural matching is more generally applicable than non-structural matching. In this case, the hypothesis of bijectivity between correspondent elements does not yield, and thus correspondence can also depend on the neighbourhood of an element. Hence, the weaving metamodel must be able to capture the possible transformation patterns belonging to the identity. In order to do so, [Del Fabro, 2009] proposed a weaving metamodel describing the semantic links between two sets of models, illustrated in Figure 10-8. Since this meta-model is grounded on the MOF meta-metamodel, a new weaving meta-model should be designed in non-MOF contexts. This metamodel only allows the simplest correspondence: associating any number of elements from (potentially) different metamodels. It only can be used for matching, since it does not precise which operation must be performed on update of either side, neither does it allow managing conflicts between correspondences. This metamodel includes the following meta-classes:

- *Wlinks* are the most basic matching or composition link, associating multiple endpoints *WlinkEnd*;
- *WlinkEnd* indicates the type of elements being composed ;
- *WelementRef* points to the elements of the input metamodels ;
- *Wmodel* is the root container for the model composition rules.

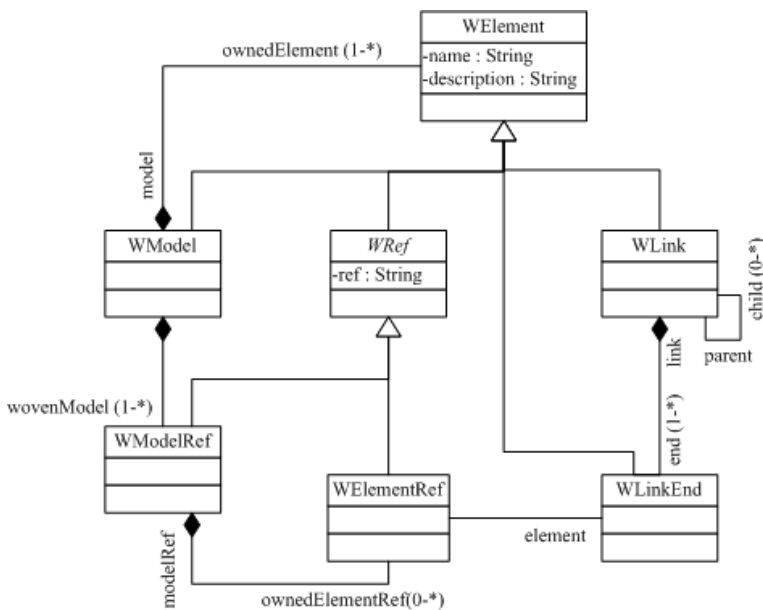


Figure 10-8: AMW Core Metamodel

The *Wlink* meta-class is then extended for convenience in specialized classes describing their multiplicity: many-to-many, many-to-one and one-to-many (raw *Wlink* are used to describe one-to-one multiplicity).

10.1.2.2.1.2 Modelling metamodels heterogeneities

Although the kind of metamodel described in Figure 10-8 allows describing semantically-aligned metamodels overlap, it does not allow expressing less aligned metamodels heterogeneities. It is impossible, for instance, to describe a mutual exclusion between two elements of two different meta-models. Additionally to the core metamodel described above, the authors of the AWM designed an extension allowing describing five common correspondence patterns:

- **Equivalence**, indicating a similarity, yet not exactly the same value. This relation is associated with a similarity value ;
- **Disjointness**, indicating that two elements from different meta-models are mutually exclusive ;

- **Generality**, indicating than one metamodel is more general than the other ;
- **Non equivalence**, denoting that an element must not be mapped to any element of the other model.

10.1.2.2.1.3 Intermediary meta-models

Other tools such as the open-source tool OpenFlexo² require the user to design an intermediate meta-model, whose elements are linked to sources and destinations models. This approach makes the maintenance of the correspondences easier, since the overlap is formally defined. Figure 10-9 illustrates this approach with two really simple metamodels. In the middle, an intermediate metamodel is designed. Since both source metamodels (left and right metamodels) are semantically near, the intermediate metamodel is as large as their intersection.

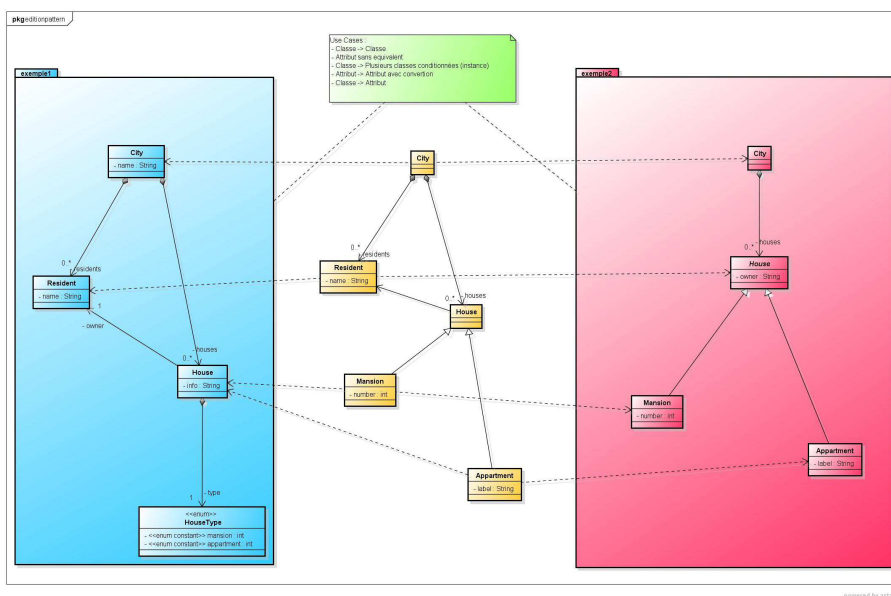


Figure 10-9: OpenFlexo intermediate metamodel

Designing an intermediate metamodel allows capturing the semantic relations between the different metamodels. It is a solution quite easy to maintain, as it does not describe complex relations. The fact that the intermediate metamodel is a legitimate viewpoint on the metamodel also points it out as a subject of analysis. Furthermore, since the metamodel potentially conforms to the same meta-metamodel than sources metamodels, it can be designed by any domain expert.

The intermediate metamodel approach, as suggested by the given example, leads to exploding complexity in cases of large and semantically aligned metamodels. The issue becomes particularly severe when there is more than two meta-models to map. This issue could however be addressed by pairing metamodels (intersections between each pairs of metamodels) instead of a global metamodel that would be the intersection of all the metamodels.

Another drawback of the intermediate metamodeling approach is linked to its relative simplicity: the intermediate mapping single is not expressive enough to express most correspondences. For instance, no restriction can be expressed on the correspondences between to linked elements of the source metamodels – the elements are either always bounded or never. Another issue is that, depending on the used technology, relations on elements could not be put to correspondence (this is the case in the OpenFlexo tool). Such conditions only hold on really well-aligned metamodels. Thus, intermediate metamodeling can only be done in (rare) well-aligned models, or should be helped with a regular, structural matching language.

² <http://openflexo.org/tiki/Welcometo+Openflexo>

10.1.2.3 Combination and replacement rules

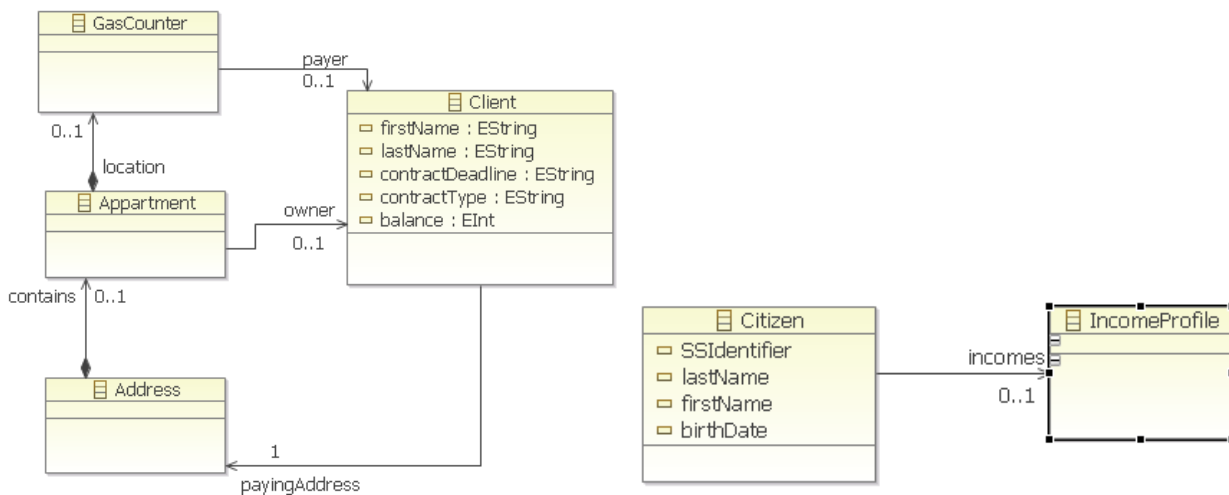
Once that correspondences at meta-model level have been defined, a tool must proceed to the application to each matching instance. One classical method is to generate transformation rules. This approach is adopted by AMW, which generates either ATL or XSLT transformation rules for the source models and delegate the work to dedicated tools. Another approach is to generate manually the different transformations in a without going with a few dedicated instructions, sacrificing genericity for efficiency. The latter technique is presented in this section. Transformation rules through existing transformation languages and tools are covered in chapter 10.1.2.6.

10.1.2.3.1 Combination

Combination rules have long been theoretically studied and implemented, since the works of IBM around HyperJ.

There are multiple strategies for performing combination between two metamodels, but the *merge strategy* is most often referred to. Merging consists in building the union of their instances and then fusing instances from *equals* classes. At a single class level, properties must also be merged in the same fashion. To put it simply, the merge operation is a union at instance-level from a set point-of-view.

Figure 10-10 illustrates the merging of two classes (Client and Citizen) from two different metamodels (the first one representing gas distribution, the second one representing tax computation). Three cases appears in the schema: common properties (both *firstName* and *lastName*), properties owned by only one class (for instance *balance* in the first metamodel, and *birthDate* in the second one), and reference to other classes. The problem of ownership is not present in this example. As expected, the resulting metamodel is the union of the two metamodels. This merged metamodel is only conceptual, since in practice the merging is done at model level, not at metamodel level.



Merge (Client, Citizen)



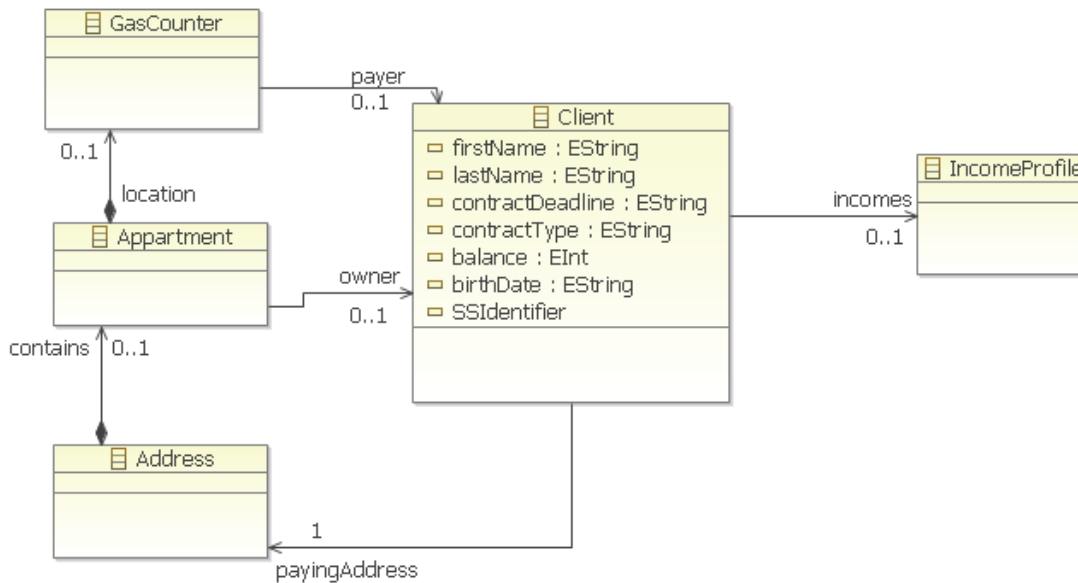


Figure 10-10: Merging at metamodel level

In trivial cases, merged elements are the same as matched elements. It is however not always the case, since the merging of a set of elements can depend on their neighboring and, more generally, be dependent on their respective positions in the model graph. Thus, combination must be independent from correspondence. Most tools offer distinct languages for expressing correspondence (*matching*) and combination (*transformation*).

In an example more relevant for model synchronization, only properties and associations present in both source meta-models would be merged. In another words, the merge operation would be done twice and results be respectively conforming to each source meta-model.

10.1.2.3.1.1 Optimization

Whatever the solution elected, transformations must be applied regularly, particularly in case of incremental synchronization on concurrently edited models. Since transformations are quite expensive to apply on large models, optimization techniques are presented.

10.1.2.3.1.1.1 Matching-level optimization: incremental pattern matcher

A first level of optimization can be done at matching time. A naïve implementation of matching will check all transformation rules left-hand-side against the whole model in arbitrary order. Incremental pattern matching like Rete-algorithm class of optimizations builds a digital tree indexing hierarchically the transformations rules and referring the matching patterns. The path from the root node to a leaf node defines a complete left-hand-side rule. With this approach, all occurrence of a pattern are stored and thus can be matched in constant time. After all matchings, the Rete network is updated accordingly to performed actions (new patching patterns are added to nodes list of references while patterns no more matching are no more referenced). The Rete network approach was used in VIATRA2.

Conceptually, Rete networks work similarly to relational algebra. Instances and relations are handled as tuples while classes and associations are associated to schemas. A partially matched pattern is stored as a specific type of tuple. Two individual matched patterns can be joined together in order to build a pattern matching to the association of both patterns. Note that the new pattern will reference the original patterns, not copy them. Thus, each pattern is only matched a single time even if it is presents in multiple larger patterns. Intermediate tuples from join operations are also stored as matched patterns. In case of update, the newly added or removed element is added as a tuple into corresponding table, and connected partial and final matches tables are recomputed. Figure 10-11 illustrates the organization of the Rete network for a

simple Petri net in the VIATRA2 tool [Bergmann, 2008]. The rule simply matches non-empty, non-final places.

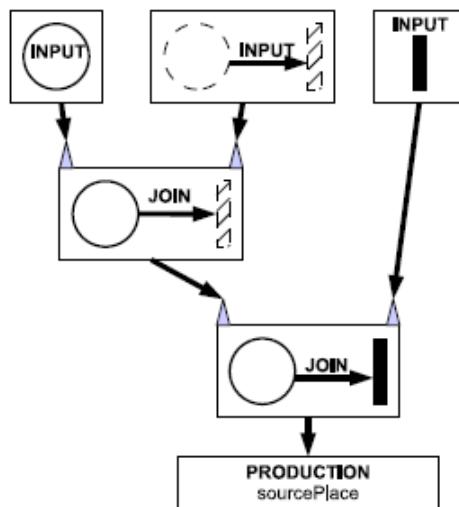


Figure 10-11: Example of VIATRA Rete network

Incremental pattern matching does, however, imply a significant increase in memory footprint as well as a more expensive initial matching phase, since the index must be built. More generally, the efficiency of incremental pattern matcher approach is dependent on whether the models are connected or not. Typically, large batches of updates imply heavy modifications on the digital tree and thus are very expensive. On model sets that are manipulation-intensive rather than matching-intensive, this approach could not be optimal.

10.1.2.3.1.2 Data value expression

An important issue of in industrial correspondences is the definition of associated attributes values. Data value expression differs from mapping expression since they also evaluate the model elements, not only the metamodel elements. A property value can be a complex association of related properties in another model, and must be defined as a regular expression.

This issue can be easily resolved with an interrogation and manipulation language on the metamodel. The OMG published a standard for performing such tasks on any MOF-compliant metamodel: the Object Constraint Language (OCL³). OCL is fully implemented in an open-source module of the Eclipse framework.

Another fitting candidate is the OMG MARTE⁴ language which defines a Value Specification Language (VSL) allowing computing expressions on an element's property. Comparatively to OCL, it supports natively complex data structures such as range, allowing to write expression independently to the actual implementation of these structures (for instance, in the range example, it could be implemented either with two independent properties *min* and *max*, or with an array of size two, amongst other solutions).

10.1.2.4 Correspondences in data bases

The problem of establishing correspondences between data bases schema (description of the data bases tables' organization, corresponding to the meta-model in the MDE context) arises in multiple contexts, one of the most important being data warehouse building, evoked in the next chapters.

³ <http://www.omg.org/spec/OCL/>

⁴ <http://www.omgmarTE.org/>

However another use exists for correspondences: the *view update problem* is the common term used to describe the study of related the relationship of changes in a *view* (in the relational means of the term) to changes in the database model. In this particular case, S and T (respectively source and target models) are known, as well as Q (the query), the unidirectional function from S to T. The view update problem lies in finding Q' (called the *translator* of Q), the unidirectional function from T to S. The issue addressed is thus not establishing links or correspondences between models elements' (since those correspondences already exist under the form of the query), but reversing a transformation operation.

Since the problem is nearly as old as relation databases (being reported as early as 1974 in [Codd, 1974]), it has received numerous responses, which are described in [Chen, 2011].

- “Clean source” approach, which is a mapping source exclusive to the target element considered. An update can be reversed only if it can be reduced to clean sources without involving other objects. This approach is used to establish updatability of a view rather than performing the actual update. It is too restrictive to be practicable in many cases ;
- Consistent views approach, which considers respective properties of the update on view and the corresponding update on database, particularly determinism. A specialization of this approach is the view complement approach, restricting the source target to the meta-elements actually represented in the view ;
- Semantic approach was first advocated by [Keller, 1986]. It consist in establishing a theoretical *view definition facility* which builds translator according to questions asked to the view builder, at view definition time, with reasonable restrictions on the updates in the context of relational databases. An example of implementation is presented in [Sheth, 1988].
- CSP approach, proposed in [Shu, 2000], advocate the transformation of the view update problem into a constraint solving problem, by considering both the view and the view update request. This method solve a large range of updates and relies on well-known techniques, but it must be run at instance level, not at schema level, and at update time contrary to the other methods that apply at view definition time. Hence, it is likely to involve serious performance issue, especially in cases where updates occurs frequently.

These approaches offer interesting insight to reverse unidirectional operations (or to exploit unidirectional mapping in order to perform back-transformation). However, the relation between a view and a source model is quite specific in relational databases, since the view is always a subset of the source model. Thus, the pre-cited approaches should be studied with this consideration in mind.

Although the approach is more methodological than technological, the semantic exploration approach initiated by Keller, in particular, is an interesting lead on how correspondences should be built and which information should they contain.

10.1.2.5 Synchronizing heterogeneous models

According to the ISO/IEC 42010 norm [ISO42010], an architecture description is mainly composed of *views* and *correspondence rules*. In the standard acceptance, a view is itself composed of a set of models. Note that the standard does not make assumption on whether the different models instantiate different metamodels or a single one. In another words, it does not explicitly address the issue of heterogeneous models.

In the case of heterogeneous models, semantical overlaps between the different models cannot be avoided in a realistic architecture. The ISO 42010 however describes the *correspondence* mechanism, allowing to define mapping between two or more architecture elements. This mechanism is mainly used a semantical connection between two different models⁵ (although other usage are allowed by the architecture description metamodel and very little explanation is explicitly given in how the mechanism should be used).

The correspondence rules are an upper-level connection governing a set of correspondences. It is an atomically-verifiable set of correspondences, set together for checking consistency concerns.

⁵ “Correspondences and correspondence rules, as specified in 5.7.2 and 5.7.3, may be used to express, record, enforce and analyze consistency between models, views and other AD elements within an architecture description”, ISO/IEC FCD 42010, p. 17, 2010

From a concrete, design-oriented viewpoint, correspondence must be enforced after edition. This action is named *model synchronization*.

10.1.2.6 Transformation rules

To perform model synchronization, a possible technique is the use of transformation rules defined in transformation language. Although model synchronization is not the primary usage of transformation language (originally though in a more generative context), they allow defining transformation rules between two meta-models and offers an engine inferring the actual transformation from the corresponding source models to the corresponding destination models.

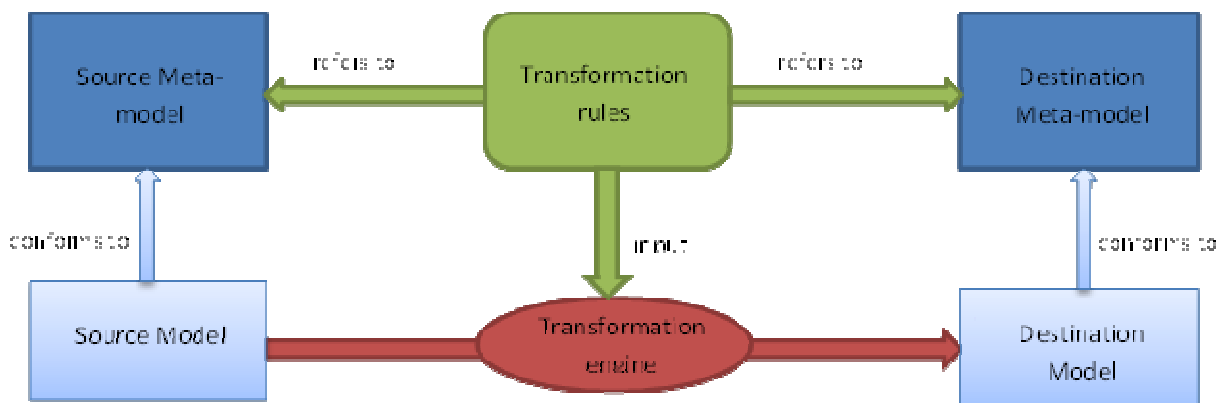


Figure 10-12: Transformation rules usage

Figure 10-12 illustrates the relation between transformation rules, source and destination metamodels and source and destination models. This relation displays the general case of model transformation. In the specific case of model synchronization, the distinction between the so-called “source” and “destination” models (and metamodels) does not yield. This raises specific issues like the need of bidirectionality for rules, cyclic correspondences and how to handle non-overlapping elements.

The purpose of the present chapter is to describe the qualities of the main transformation languages and engines in performing model synchronization.

10.1.2.7 Transformation properties for model synchronization

The expressivity of transformation languages is quite variable, and is a critical capacity for model synchronization. Czarnecki and Helsen [Czarnecki, 2003] defined a taxonomy of graph transformation languages properties using domain analysis. Each rule is described with a feature model and can be composed of many features which are not described in the present document.

- **Specification** defines dedicated specification mechanism such as pre/post conditions.
- **Transformation rules**, which describes the expressivity of the rules themselves, such as parametrization capacities, possibility to declare intermediate structures, typing, syntax, etc.
- **Rule application control** defines *location determination* - the strategy used in order to apply a given transformation rule to the different matching parts of the source model, which can be either non-deterministic, deterministic or interactive, and *scheduling* – the order of application of the matching transformation rules. It can be either implicit in the language, or explicitly stated by the user.
- **Rule organization** defines the rules of composition of individual transformation rules, such as packaging them into modules, use of reuse mechanisms such as inheritance or templates, as well as the organizational structure defining whether the rules are described following the structure of the source, the target or neither.
- **Source-target relationship** sets whether the source model is identical (in case of in-place transformations) or different from target model.

- **Incrementality** describes the capacity to update, partially or not (some languages only allow additive updates), the target model.
- **Directionality** describes the capacity (or lack of) of a transformation language to be bidirectional – i.e., the source and target models can be inverted (while executing the rules from right-side to left-side).
- **Tracing** describes the support for traceability links between source and target. This support can be stored in source or target models or in a separate location. It can be manually specified by user, or automatically computed.

10.1.2.8 Model-to-model transformation taxonomy

These properties were designed to be applied to any transformation, either from model to model or model to code, to text, etc. In the case of model synchronization-oriented transformation languages, only model to model transformations are relevant.

Furthermore, language features are only a part of the features that must be investigated: in our concern, the framework characteristics are also relevant.

It is useful to divide the features between language and execution engine. We thus propose a reinterpretation of Czarnecki's features in the context of model-to-model transformation:

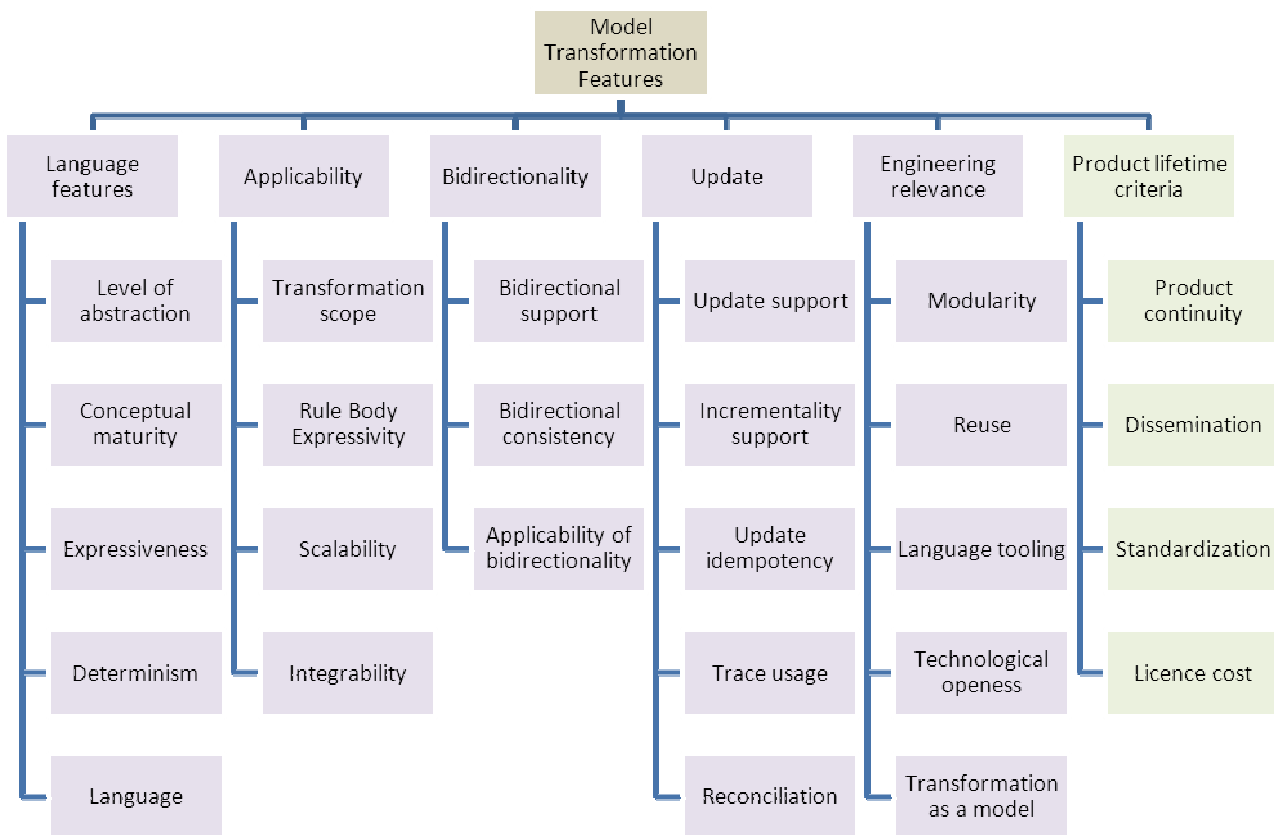


Figure 10-13: Model-to-model transformation features

10.1.2.8.1 Language features

Level of abstraction

Transformation rules can be either fully declarative – which involves a notion of *mapping* between source and target – imperative or hybrid. The last case corresponds to a declarative framework completed with an action language describing how to perform the transformation. A mapping, in this context, is a declaratively-

defined relation between the features (elements, reference values, attribute values) of source and target models.

Note that for the style to be truly declarative, mapping specifications must never rely on an implicit or explicit notion of *state* of the target models during the transformation process. A state, in this case, would be a “current content” of the target models and/or the value of non-constant temporary variables.

The abstraction level also define the semantic alignment between the rule description language and the models.

Conceptual maturity

Conceptual maturity defines whether the transformation concepts are clear and orthogonal, and if a methodology has been described for their usage.

Expressiveness

Expressiveness defines the scope of the transformations that can be defined with the technology. In other words, the range of transformation that cannot be defined because of conceptual restrictions.

Note that very often, this information, although essential, is not available. In that case, we expose the limit we found in our experiments on use cases.

Determinism

Non-determinism in terms of the result of the transformation is not acceptable. If the technology allows non-determinism, this criterion defines whether it is easy to resolve or forbid it.

Language.

Some transformation technologies impose a textual language, while some other also offers a graphical language, generally tooled with a graphical editor. In the first case, it is necessary to describe whether the language supports escaping to a mainstream general-purpose programming language (GPL) such as Java.

What is at stakes is: a) cultural habits: using a known GPL vs. learning a new DSL, b) performance tuning by the user via a low-level GPL, c) access to a mature library for data manipulation.

10.1.2.8.2 Applicability

Transformation scopes

Some technologies impose restrictions on the physical or logical scope of the source and target:

- Physically, by imposing to store models in files and excluding databases, or by allowing only one target/source file ;
- Logically, by allowing only one source/target “model”, and/or forbidding to specify the source/target be a subset of a model.

This criterion defines which restrictions in scope (if any) the language and framework involves.

Possible case studies: transform a fragmented model (several files, several Epackages), transform a Team model (database not file), transform from/to a model and an associated Viewpoint resource (e.g., Safety) or pattern catalogue.

Scalability

Defines how do transformation technology behaves in processing time and memory usage when model size grows. As an indication of realistic size, large-size models in industry are about 200,000 model elements.

Version	Nature	Date	Page
V01.00	R	2014-04-2830	82 of 117

Integrability

Defines whether a transformation can be executed programmatically from an Eclipse environment. I.e., is there a clear Java API for configuring, starting and controlling (progress indication, cancellation) a transformation?

10.1.2.8.3 Bidirectionality

Bidirectionality support

Defines whether a transformation and its reverse transformation can share code. The alternative is that reverse transformations must be written from scratch.

Bidirectional consistency

Defines what mechanism (if any) exists to guarantee that a transformation and its reverse are consistent.

Applicability of bidirectionality

If the technology supports bidirectionality, defines how easy is it to determine in which cases it is possible, or, said otherwise, what is the expressiveness of the bidirectional sublanguage.

10.1.2.8.4 Update

Update support

Defines whether the technology is capable of modifying existing target models according to changes in source models.

Incrementality support

Defines whether the technology is capable of only using the modified part of the source model and propagate those changes to the target. Note that support for incrementality implies support for update.

Also include the support of delayed incrementality, i.e. if it is possible to update the source model while the target model is not “on-line” and how the target is updated latter in this case.

Update idempotency

In the context of model transformation, idempotency is the capacity to *not* updating a target model when a given transformation is performed multiple times in source model. This feature is generally implemented by using traces and difficult to get otherwise.

This property is essential for the definition of bridges between modelling tools.

Trace usage

The engine keeps track of the links between source and target model elements. Tracing can be automatically generated after a first transformation and then maintained after each synchronization. Also includes the possibility to create manually traces from pre-existing source and target models.

Reconciliation

Is it possible to specify precisely what is updated when both the source and target models have changed before the update? Is it possible to let the end-user decide in certain situations (ambiguous cases or “conflicts”)?

Version	Nature	Date	Page
V01.00	R	2014-04-2830	83 of 117

10.1.2.8.5 Engineering relevance

Support for modularity

Defines whether it is possible to package rules into modules and if these modules can compose (i.e. import) other modules.

Support for reuse

Defines whether it is possible to reuse or derive individual rules from other rules. This covers several mechanisms from rule composition through calling to rule inheritance.

Language tooling

Defines whether the transformation technology imposes a textual language and what is the maturity of its tool support (e.g. editor, debugger...).

Technological openness

Can third-party technologies for model manipulation/querying be used for defining a transformation? E.g., query technologies such as Eclipse-OCL, EMF-incQuery, Acceleo; model description technologies such as HUTN.

Transformation as a model.

Reasoning about transformation is a very important feature, since transformation rules are likely to come from different sources and can be redundant or even contradictory. Furthermore, it is possible to generate code from high-level transformations or even transform a set of transformation (like it is done for requests in a database context, for instance). Formalizing the transformation through models allows such reasoning.

10.1.2.8.6 Product Lifecycle Criteria

Product Continuity

Describes how frequently the technology is updated and if there is regular stable releases.

Dissemination

Describes the size and activity of the community around the tool (forum, mailing lists, etc.). This information which may reflect its level of maturity, and can also reveal who offers alternative and free support, especially in open source projects.

Standardization

Defines whether the technology relies on existing standards or is likely to lead to new standards.

License/cost

Describes which licence and what is the cost of the technology usage is.

10.1.2.9 Transformation languages

In this section, a review of relevant transformation languages and engines is done with respect to the criteria described in previous sections.

10.1.2.9.1 ATL

The AltanMod Transformation Language (ATL) is a unidirectional transformation language developed by the ATLAS Group (INRIA & LINA) within the Eclipse framework. It has both declarative and imperative style

Version	Nature	Date	Page
V01.00	R	2014-04-2830	84 of 117

transformations. It forbids providing the same model as input and output, making it a language non-recursive by construction. Transformations rules are specified in *modules*, which contain a *header* and an *import* section, as well as multiple *helpers* and *transformation rules* sections.

A *helper* is reusable code and applies either on *operations* or *attributes*. *Operation helpers* are reusable function definitions, taking elements and context of the source model as parameters. *Attribute helpers* allow associating constant values with elements of the source model. *Transformation rules* express the actual model transformation. Rules can be either *called* (imperative) or *matched* (declarative). The matched rule contains optional action block, which can trigger called rules. Finally, ATL offers a large range of types, from primitives (string, numerical, etc.) to tuples. These types are very close to OCL types. Figure 10-14 illustrates a very simple ATL rule, which translates persistent classes into tables of the same name into a target model of kind RDBMS.

When a rule is successfully matched on the source models, the output model is created, and a traceability link is automatically created. A rule can be either *standard* (applied only once for every match), *lazy* (called by other rules, and applied on a given match as many times as called by caller rules) and *unique lazy* (call by other rules and only applied once by match). These policies allow to perform some basic management of concurrent transformations.

```
rule PersistentClass2Table{
  from
    c : SimpleClass !Class (c.is_persistent)
  using
    nm : String = c.name + '_table';
  to
    t : SimpleRDBMS!Table (name <- nm)
}
```

Figure 10-14: ATL transformation rule

Generally, ATL supports multiple inputs and outputs models. These models, however, must be different. In order to implement in-place transformation, ATL offers a refinement mode in which input and output models are the same. This mode does not support the lazy (or unique lazy) policies, thus impeding any rule application scheduling in this case. Furthermore, it is not possible to make a partly endogenous transformation, i.e. to modify a (single) model within a set of models, if the set contains more than one model.

Concerning heterogeneity, ATL is built upon a pivot metamodel which offers connectors to many other formalisms like EMF/Ecore, KM3, XML, etc.

As a partial conclusion, ATL does allow to perform quite complex transformations, yet enforce quite drastic restrictions. It could be argued that partial in-place transformation, although not necessarily needed in model synchronization, should be a desirable feature. It is quite likely that deciding algorithms in a concurrent edition context would use this method. More generally, it seriously impedes incrementality. Furthermore, ATL rules being non-reversible when action blocks are involved, using them in a necessarily two-way process would imply to double the number of rules to write and maintain.

Another major issue with ATL is the lack of expressivity of the source pattern part (*from* section). It does not allow expressing condition addressing more than one source elements, which force to put the multiple-arguments constraints in the imperative action part, thus reducing the reversibility of the rule.

	Language Features					Applicability	
	Abstraction level	Maturity	Determinism	Language	Scope	Scalability	Integrability
ATL	Quite high when declarative, same as OCL in imperative style	TRL8	Y	Textual	Multiple inputs, multiple outputs	Inherited from Eclipse/EMF environment	Eclipse environment, programming API

	Bidirectionality			Update features				
	Bidirectionality Support	Consistency	Applicability	Update support	Incrementality support	Update idempotency	Trace usage	Reconciliation
ATL	N	NA	NA	N	NA	NA	Y, automatic	N

	Engineering relevance					Product lifecycle			
	Modularity	Reuse	Language tooling	Openness	Transfo as model	Product continuity	Dissemination	Standardization	Licence/cost
ATL	Libraries, modules	Superimposition, rule inheritance	IDE, debugger	Can use either OCL constraints or Java helpers	Y	Lively project: Last release : 2013/05, one release each 6 month.	Excellent, ATL is the most common transformation language on the market	Loosely adapted from OMG QVT, use an obsolete and loose version of OCL	EPL licence

10.1.2.9.2 Medini-QVT

QVT Relational [OMG, 2005] is the OMG's proposed language for declarative model-to-model transformations, inspired from relational algebra. It belongs to a set of transformational languages which also allow imperative transformation specifications (with Operational Language), and conform to explicitly stated metamodels. The first version of the standard was released in 2005.

The language allows either single-direction or bidirectional transformations, as establishing relationship between pre-existing metamodels as well as incrementality (either destructive or additive updates).

Endogenous transformations are supported in a multi-model context. Transformations are defined by a non-empty set of models, associated to a mode (*in*, *out*, *inout*), offering the greater flexibility in domain definition. Conditions can be stated for models acceptability, with *when/where* clauses.

A relation is a constraint that must be satisfied by the elements of the candidate models. It is defined by a set of domains and a pair of *when* and *where* predicates.

- A *domain* is a typed sub-graph from a model described through a pattern ;
- The *when* clause describes an arbitrary condition on the model that must be verified (i.e. a precondition for the transformation). If this condition contains a QVT relation, this relation is verified (hence the target domains are potentially modified) before the execution of the *where* clause ;
- The *where* clause describes a condition applicable on all element of a model. It will perform the corresponding transformation on target domains.

```

relation ClassToTable
{
    checkonly domain uml c:Class {
        namespace = p:Package {},
        kind='Persistent',
        name=cn
    }
    enforced domain rdbms t:Table {
        schema = s:Schema {},
        name=cn,
        column = cl:Column {
            name=cn+'_tid',
            type='NUMBER'},
        primaryKey = k:PrimaryKey {
            name=cn+'_pk',
            column=cl}
    }
    when { PackageToSchema(p, s); }
    where { AttributeToColumn(c, t); }
}

```

Figure 10-15: A relation in QVTR

Both *when* clauses and *where* clauses are standard OCL expressions that support calls to QVT relation as leaves.

The base mechanism of QVT is target-model matching: if a relation is not verified between a set of (at least two) domains, the elements are built in the target domain in order to make the relation true. QVT can also be used in order to check the relation on a model, without transformation implied. In order to select the mode of the relation, modes are assigned to domains (either *enforced*, which allows modification, *checkonly*, which forbids it, or none). Bi-directional transformation will have at least a domain in each model marked as *enforced*. Note that this condition is not enough to define a bi-directional transformation: according to the mode of the corresponding model (in, out, or inout), the transformation will have different semantics. For instance, if both source and destination models are enforced, not only matching patterns in source will create non-already existing patterns in target, but matching targets in target not related to matching sources will be deleted. As one can see, the check-before-enforce policy of QVT is far from trivial.

The two-level enforcement defined by the *when/where* clauses does present some risks: since the order of execution of the relation defined in the *when* clause is not defined, the relation defined in the *where* clause can be matched against different domains and thus inducing non-determinism in the transformation, as outlined by [Stevens, 2007]. A solution would be to perform global constraint solving (or any other exploratory solution) in order to find the “better” possible match, yet both efficiency and, more importantly, pertinence of this approach would be questionable – because to define a priori a “better” criterium on a transformation requires knowing the actual context of use. Note that the *when* clause, however, allows to express conditions over multiple parameters, what ATL’s source pattern mechanism does not allow.

Figure 10-15 illustrates an example of relation mapping UML classes to databases schemas. This relation states that, if (*when*) a package and a schema verify a certain relationship *PackageToSchema*, then (*where*) the relationship *AttributeToColumn* must also be verified. The schema will be modified in order to enforce the relation if necessary.

Although there is multiple implementations of QVT Operational (e.g. SmartQVT, EclipseM2M), QVT Relational is only implemented in Medini-QVT⁶, which supports any EMF models. It does automatically generate traces. However, it does rely strongly on those traces, and does not perform a target-model

⁶ <http://projects.ikv.de/qvt/>

matching on pre-existing targets, contrary to the QVTR specification. This limitation severely impedes incrementality, since it implies that the target model must not evolve on its own.

Another limiting factor is the lack of management apart from the dependency graph. Transformations rules application is implicitly supposed to be sequential. Concurrent transformations are not handled in a deterministic manner. How the successive transformations order impacts on the model is not analyzed nor managed in our knowledge.

Actually, QVT in general and its Medini implementation is though primarily as a bidirectional transformation language rather than a synchronization language. Hence its canonical usage consists in generating a model from another one, and potentially reversing the transformation if needed. Although some trace-based support for incrementality is provided, the transformation is meant to update solely the original target. Neither the engine nor the language was thought to be used in a concurrent context.

	Language Features					Applicability	
	Abstraction level	Maturity	Determinism	Language	Scope	Scalability	Integrability
QVTR	High abstraction level	TRL8	Y	Textual	Multiple inputs, multiple outputs	Inherited from Eclipse/EMF	Eclipse environment, programming API

	Bidirectionality			Update features				
	Bidirectionality Support	Consistency	Applicability	Update support	Incrementality support	Update idempotency	Trace usage	Reconciliation
QVTR	Y	Y, through double transformation	Can be expressed through preconditions and pattern matching	Y	Partial, relying on traces and thus does not support target evolution. Offline.	Y	Y, automatic	N

	Engineering relevance					Product lifecycle			
	Modularity	Reuse	Language tooling	Openness	Transfo as model	Product continuity	Dissemination	Standardization	Licence/cost
QVTR	Libraries, modules	Superimposition, rule inheritance	IDE, debugger	Can use OCL constraints	Y	Closed project: Last release: 2011/03, dead forum for more than 6 month.	QVT Relational is an OMG norm	Loosely adapted from OMG QVT, use an obsolete and loose version of OCL	EPL licence for non-commercial use, Commercial use to be negotiated with <i>ikv++ technologies</i>

10.1.2.9.3 AToM

AToM⁷ (A Tool for Multi-formalism and Meta-Modelling) is a multi-paradigm modelling tool, primarily aiming to transforming models from a meta-model to another one. Both models and meta-models are described by graphs, while transformations are described by graph-grammars models. The transformation syntax is fully declarative.

While one of the objectives of AToM is model transformation from a meta-model to another one, Each transformation is defined by a rule which is composed of two sides. The left-hand side defines a pattern that is matched at run-time against the input model. For each match found, it is replaced by the pattern defined in the right-hand side of the rule. Thus the tool allows graph rewriting rather than graph transformation, restricting itself to endogenous transformations. No support is offered to incrementality and no tracing is done since the original model is transformed.

The rules themselves support a basic notion of scheduling (*order* attribute). Rules' metadata can also be explored at runtime, allowing to determine whether a rule as already been triggered, thus addressing unwanted recursion issues. Left-hand and right-hand are edited separately and pre-condition and post-condition can be expressed in OCL. Rules can call other rules. Sound formal foundations of graph grammar

⁷ <http://atom3.cs.mcgill.ca/>

may assist in proving transformation correctness and convergence properties – yet such functionalities must be developed by the user.

An important issue of AToM is that it enforces a proprietary meta-metamodel for the different formalisms to be synchronized. In the context of model synchronization, it would mean to redesign both source and target formalisms (and models) in order to make them compliant with the AToM framework. This approach is clearly unrealistic in actual industry projects.

As a partial conclusion, although AToM has some interesting features in terms of rule application scheduling, its scope is not adapted to perform model synchronization.

	Language Features					Applicability	
	Abstraction level	Maturity	Determinism	Language	Scope	Scalability	Integrability
AToM	High abstraction level (3-graph grammar)	TRL8	Y	Graphical	Single input, single output	Unknwon	Stand-alone, does not support EMF models

	Bidirectionality			Update features				
	Bidirectionality Support	Consistency	Applicability	Update support	Incrementality support	Update idempotency	Trace usage	Reconciliation
AToM	N	NA	NA	N	N	NA	N	N

	Engineering relevance					Product lifecycle			
	Modularity	Reuse	Language tooling	Openness	Transfo as model	Product continuity	Dissemination	Standardization	Licence/cost
Atom	N	Unclear	Graphical editor, linear solver, code simulator and generator	Can use OCL constraints	Y	Closed project: Last release: 2007/02.	Limited academic diffusion only	No	Open source

10.1.2.9.4 VIATRA2

VIATRA (Visual Automated model TRAnsformation) is a model transformation framework aiming for model analysis and transformation, either to code or to other formalism. In order to do so, it combines graph transformations and abstract state machines.

Graph transformations are used to define elementary transformations, combined through state machines rules. Thus it does allow a good level of rule application scheduling. VIATRA also offers the possibility to parametrize the transformation by passing types as parameters, thus allowing the design of transformation templates. This last feature is unique in transformation language. Transformation rules are expressed into a specific language, VTCL, which combines declarative features (graph transformations rules) and imperative feature (state machines implementing control flow). It is also possible to plug arbitrary Java code, at the cost of rule's reversibility.

Left-hand side is always defined by a pattern, while right-hand side can be defined either by a pattern or by an action (latter case illustrated in Figure 10-16).

Patterns are non-injective by default, but the *shareable* keyword allows injectivity (i.e. a pattern variable can be only matched against more than one element in the model). The different layers of caller rules apply successively their policies, which means that if any one rule is non-injective, the top-level return will not return twins values either. Negative (rule-disabling) patterns can also be defined.

VIATRA2 engine relies on Rete network in order to determine offline an efficient scheduling for the different rules constituting the transformation. An optimal graph of *constraints* (elementary parts from a pattern) that matches against the actual graph is created. Hence, modification of the source model will force recreation of the graph, but generation from a (relatively) stable source model will be optimally efficient.

```

machine PM
{
    pattern connectedNodes(N1,N2) =
    {
        entity(N1);
        entity(N2);
        relation(R,N1,N2);
    }
    rule main() = seq
    {
        forall Src,Trg with find connectedNodes(Src,Trg) do seq
        {
            println(fqn(Src) + " -> " + fqn(Trg));
            let NewR = undef in seq
            {
                new(relation(NewR,Trg,Src));
                rename(NewR,"reverse");
            }
        }
    }
}

```

Figure 10-16: VCTL transformation (create a reverse connection)

[Bergmann, 2009] presents results from precise benchmarking of VIATRA2. The result was quite good, since it displayed linear growth of the model memory footprint (including overhead due to the Rete network), and VIATRA2 could manage up model spaces up to 1.5 GB (about 100000 simple elements). Amongst other contexts, a benchmark for model synchronization was tested. As for execution time, with quite heavy constraints (from 20% to 50% of a total of 30000 elements are modified, deleted or renamed), results were also satisfactory, with a linear growth of both start-up and synchronization durations and a WCET of about 10000 ms. Although 30000 elements is about one order of magnitude less than currently existing industry models, it leaves space for further improvements. Others benchmarks can be found in the tool-dedicated webpage⁸.

However, VIATRA2 is not a bi-directional language, nor it allows addressing multiple models in input or output. It can be used in order to perform reverse transformation or back annotation, but implies to write reversed rules (which are not necessarily symmetrical with original rules). Note that, even in a bidirectional context, the “reverse transformation” is quite different from “two-way transformation” necessary for model synchronization. Generally, VIATRA2 is more fitted to model generation than to model synchronization.

Furthermore, VIATRA2 relies on a specific meta-metamodel (graphs) which, if it is well-grounded mathematically and would allow formal model analysis, impedes using pre-existing models or meta-models. Thus the modelling framework is quite closed and does not fit to an industrial context with many legacy models. Note that from some aspects (models and metamodels being represented in the same model space, element multi-typing...), the formalism of VIATRA2 borrows notion from the ontology field.

Note that a study to demonstrate feasibility of synchronization with VIATRA2 was done in [Ökrös, 2010]. However, the approach is largely hand-made and, like other unidirectional transformation techniques, error-prone and hardly maintainable. This study also introduces an user-made traceability model.

⁸ <http://incquery.net/performance>

	Language Features					Applicability	
	Abstraction level	Maturity	Determinism	Language	Scope	Scalability	Integrability
VIATRA2	High abstraction level (3-graph grammar and state machines)	Incubation phase (TRL 3-5)	Y	Textual	Single input, single output	Quite good, with predictable memory needs and execution time, limited to 100000 elements.	Rely on eclipse, yet does not use MOF/EMF

	Bidirectionality			Update features				
	Bidirectionality Support	Consistency	Applicability	Update support	Incrementality support	Update idempotency	Trace usage	Reconciliation
VIATRA2	N	NA	NA	Y	Y	Y	Manual	N

	Engineering relevance					Product lifecycle			
	Modularity	Reuse	Language tooling	Openness	Transfo as model	Product continuity	Dissemination	Standardization	Licence/cost
VIATRA2	N	Y, genericity through templates	Graphical editor, debugging, support for (manual) traceability edition	Can use Java code	Y	Ongoing project: last release: 2012/08.	Many contributors and users in academics	No	EPL

10.1.2.9.5 EPSILON

Epsilon is a family of integrated languages aiming to managing models [Paige, 2012]. Amongst other modules, it defines the core language (EOL), the comparison language (ECL) and the model to model transformation language (ETL).

EOL borrows most expressions from OMG OCL, while also allowing assignment, sequencing and multiple model access either in input, output or input-output modes. Figure 10-17 gives an example if EOL query searching for classes in an UML model whose name is the same than tables in a DBMS model.

```

for (class in UML!Class.allInstances()){
  if (DBMS!Table.allInstances().exists(table|table.name == class.name)) {
    (^ Found matching table for class ` + class.name).println();
  }
  else {
    (^ Not found matching table for class ` + class.name).println();
  }
}

```

Figure 10-17: EOL matching query

ETL is a declarative, single-way M2M transformation language built upon EOL. It allows preconditions as well as rule extension. It is also a mapping language, allowing to define *equivalence* between source and target models. Rules are declarative, but transformation body is imperative. When a transformation is successful, new elements are created and a trace model is updated. Figure 10-18 gives an example of a model transformation with ETL. Pre and post expressions can be defined and rules can be declared *lazy*, *greedy* (for less expensive comparisons, done replace *type-of* matching by *kind-of* matching) and *abstract*. A transformation (or its guard) can explore the source models but also the target ones, and thus may have to trigger some other rules.

```

rule SmallSquareToCircle
transform s : GeometryModel1!Square
to c : GeometryModel1!Circle {
  guard: s.size < 10
  c.radius = s.edge;
}

```

Figure 10-18: ETL M2M transformation

However, Epsilon is a generative language which, although it does handle automatically traceability, does not manage incrementality. Furthermore, only one element can be the input of each transformation, restricting the rules to surjective functions.

As a partial conclusion, although Epsilon displays interesting features in conciseness and the possibility to perform some rule application scheduling, it is not fit to perform model synchronization due to the weak expressivity of input parameters, lack of bi-directionality and incrementality. On the other hand, the framework seems more oriented toward implemented processes of kind ETL (Extract, Transform, Load) and more generally to implement life cycle management.

Epsilon is used by open-source benchmarking and safety analysis tools, developed for and by important industry actors such as aerospace (NASA Certware), railway (SAFECAP), as well as modelling or meta-modelling tools from academics or industry actors⁹.

	Language Features					Applicability	
	Abstraction level	Maturity	Determinism	Language	Scope	Scalability	Integrability
Epsilon	Low-level (imperative body rules)	Mature (TRL 9)	Y	Textual	multiple input, multiple output	Unknown	Run on Eclipse but EMF-independent. Provide a EMF connector, however.

	Bidirectionality			Update features				
	Bidirectionality Support	Consistency	Applicability	Update support	Incrementality support	Update idempotency	Trace usage	Reconciliation
Epsilon	N	NA	NA	N	N	N	Automatic	N

	Engineering relevance					Product lifecycle			
	Modularity	Reuse	Language tooling	Openness	Transfo as model	Product continuity	Dissemination	Standardization	Licence/cost
Epsilon	Y	Y, with rule extension	Graphical editor, debugging, traces, test design, workflow design, etc.	Can use Java code	Y	Live project	Both academical and industrial	No	Open source

10.1.2.10 Model transformation and MDE

Approaches and tools presented in previous sections are middle to low-level from engineering point of view. Although they generally allow modelling the transformations and offer some support to reason about the transformations rules, they are not specified in relation with formal requirements, do not offer actual analysis tools. More importantly, there is little to no support for refinement in the transformation design, and a single formalism is offered to design all levels of abstraction of the transformation. Issues like maintenance or testing are simply ignored.

Such approach is similar to the early practices of software development. However, industrial experience repeatedly concluded to the need for greater formalization of the design process. The authors of [Guerra, 2011] advocate that, transformation being software used in industrial context, with similar constraints (high complexity, long-term maintenance, need for adaptability, need for different level of abstraction and focus on specific concerns, life-cycle issues...), it should follow a similar design process.

The authors distinguish 7 features necessary in order to make the transformations compliant with engineering practices:

- Requirements traceability ;
- Analysis ;

⁹ <http://www.eclipse.org/epsilon/users/>

- Architecture design ;
- High-level design ;
- Detailed design ;
- Implementation ;
- Testing.

These requirements are managed in the TransML framework by a family of languages, as illustrated in Figure 10-19. TransML does not specify a given implementation language for the transformation, but focuses on the specification and design processes.

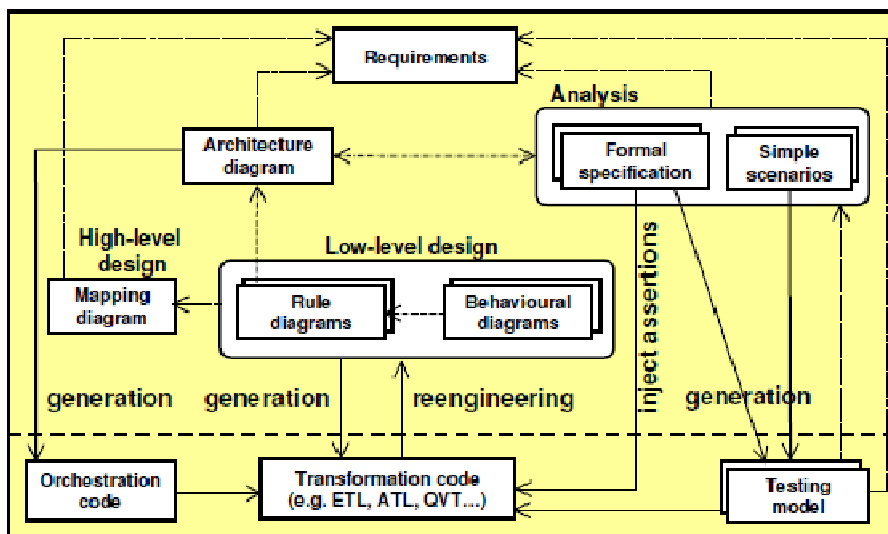


Figure 10-19: TransML transformation framework

10.1.2.10.1 Requirements and analysis

10.1.2.10.1.1 Requirements engineering

Since the knowledge of the overlap between two metamodels is generally fuzzy even for the metamodel designer, finely understanding the rationale of each transformation is a critical aspect of transformation management. Such requirements are likely to be refined and evolve during the project life-time and thus must be formalized. Maintaining traceability with further models is also a major feature for the requirements engineering.

TransML defines a tool language allowing to define requirements in a notation similar to SysML requirements diagrams, with the addition of transformation-specific notions (e.g. classification of the source of a transformation). This language supports requirement hierarchical decomposition, classification, refinement and tracability, thus allowing requirements engineering for model transformation.

10.1.2.10.1.2 Requirements analysis

In order to better understand requirements, engineers in the industry use a variety of mechanisms. Such need appears as well for requirements engineering. One of the most common mechanism is the definition of scenarios, linked with the requirements. In the TransML context, scenarios are *transformation cases* and describe different actual transformation. Transformation cases can be used to understand better the transformation at specification time and lead to a refinement of requirements and to initialize the *specification* step. They can also be used in order to feed transformation-by-example techniques (cf. section 10.1.2.11). Finally and more importantly, they are used to lead the development of test cases.

A second stage of analysis is the definition of the transformation *specifications*. Like their counter-parts in regular engineering, specification describes what the components (here the transformation) have to do, without specifying how to do it. It also allows to add pre and post conditions that must be verified. The TransML framework offers an abstract formal specification language allowing to specify either trace-based or traceless transformations.

A full traceability is maintained between requirements, transformation cases and specifications.

10.1.2.10.2 Architecture

Transformation between large models with complex overlap can imply a large number of rules. Furthermore, the transformation process is likely to be connected with other components in order to perform analysis, optimization or generation. Finally, the need for reuse of transformations is becoming more and more stringent as the complexity of transformed models is increasing. Hence it is important for the transformation rules to follow the good practices of architecting: modular decomposition and well-defined interfaces.

The TransML framework offers an architecture language allowing defining hierarchical components and connectors, and introducing transformation-specific concepts such as transformation kind (model-to-model, model-to-text, text-to-model and in-place) and ensuring by construction that inputs and outputs are consistent with the transformation kinds.

10.1.2.10.3 Design

Current transformation languages only allow to design at a limited abstraction level, generally quite low. However, it is a known fact in system engineering that multiple levels of abstraction are necessary in order to understand a system in the long run. TransML offers two levels of abstraction in order to manage the complexity of the transformation:

A *mapping model* allows to describe the transformations at a high level of abstraction. It describes relations between the elements of the different meta-models involved in the transformation (sources and targets). It only states *which* elements a transformation will impact, without stating in what way and how implementing this transformation. Mappings support many-to-many relationship and can be associated with constraints in an OCL-like fashion limiting their validity. Mapping models can be refined like any other models. They also are completely independent from the transformation implementation language.

Low-level transformation design allows to design the actual transformation, that is, *how* to do it. It is implemented in TransML under the form of two models: a *rule structure* model and a *rule behaviour* model. Rule structure will describe input and output from the transformation, when the rule behaviour will describe the operation actually performed by the transformation (e.g. attribute computation, link creation for model consistency). Rule structure and rule behaviour must of course be consistent. Both support block decomposition, offering some degree of reusability. It is to be noted, however, that low-level design is not totally independent from transformation implementation, as some operations may not be available or optimal in some transformation languages.

10.1.2.10.4 Implementing and testing

As mentioned before, implementation is not covered by the TransML framework. Languages focusing in efficiency or completeness already exist in the market (cf. section 10.1.2.9), and should be elegit according to the project priorities. The TransML framework supports ETL (from the Epsilon framework) and QVT-Relations, but authors are confident about the support to other languages like ATL.

Testing is supported in TransML by a dedicated language, which allows to describe test cases. Automatic generation of test cases is supported from *transformation cases*, or from the formal specification defined during the analysis stage.

10.1.2.10.5 Traceability between the different engineering phases

As mentioned across the previous sections, traceability is ensured all along the process, from requirements to test. Thus, it is possible during all the process to retrieve which requirement is involved, and which test

Version	Nature	Date	Page
V01.00	R	2014-04-2830	94 of 117

ultimately ensures a given architectural, specification or design element. Benefits of traceability are wide and well-studied, and apply during all the life cycle. Figure 10-20 illustrates the semantics of traces across the different models of the TransML framework.

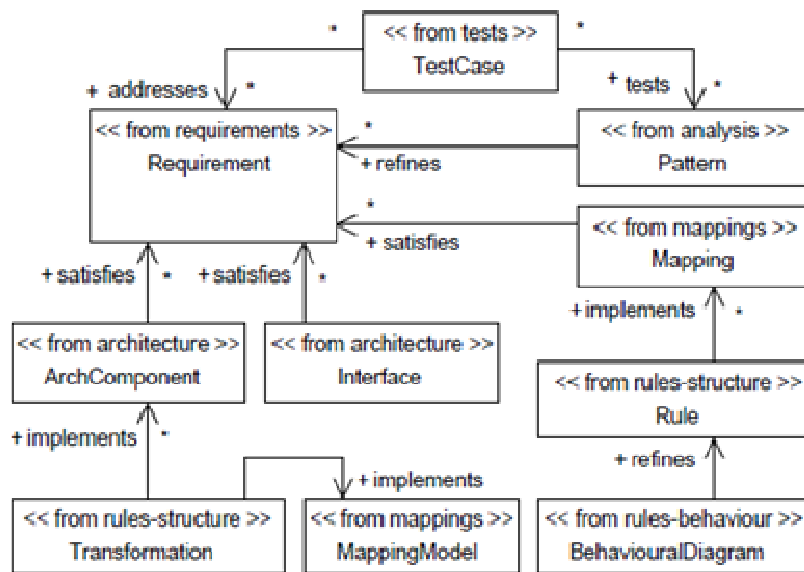


Figure 10-20: TransML framework traceability links

10.1.2.11 Semi-automating correspondence definition

Correspondence definition and representation have been explained in chapter 10.1.2. Many works, however, have been performed in order to go further and partially automatize the tedious and error-prone task of establishing correspondences. This task is often referred as *overlap detection*. There are basically two levels of overlap detection: at model level and at metamodel level.

Overlap detection at metamodel level is an issue when integrating the heterogeneous environment. In some cases, it can be enough to perform synchronization – for instance, in a fully connected environment, there is no need to automatically detect model overlap, since model traceability can be ensured by construction.

However, in more realistic cases, one cannot ensure that remote clients are connected when an element is created, and potentially similar elements can be created by multiple clients. In this case, the same techniques can be applied than in the case of metamodel alignment, although helped by the fact that the creation operation already conveys some semantics about the created items (class or type, at minima). Thus, model alignment is similar but simplest than metamodel alignment. This is why, in the next section, we only describe the latter problem.

10.1.2.11.1 Similarity-based overlap detection

The most common and simple method is to assume total overlap between matching elements, based on a matching function.

10.1.2.11.1.1 Name-based matching

The simplest form of matching function is a simple comparison between the elements identifiers, names or labels. In some case, this method is quite efficient, particularly when homogenous naming conventions were used on the different models. This hypothesis, however, does not fit with our heterogeneous models premise.

Inspired by works on ontology alignment, refinements of this method allow some tolerance to syntactic variability on the elements names.

10.1.2.11.1.2 Name-similarity matching

A common issue of name-based matching is that different domains can use different words to designate the same item, especially in case of partial overlap. An example of such difference is the word *processor* from a software viewpoint, which can be either a FPGA, CPU or ASIC.

A common solution of the limitation of name-based matching is to use a dictionary and:

- Determine true synonymous ;
- Determine generality links (e.g. both *FPGAs*, *ASICs* and *CPUs* belong to the *processor* kind).

This technique has some limits, generally shared with name-based matching: it does not allow to discriminate between homonymous concepts with different meaning according to the domain. An example of such issue is the word “function” which have different semantics in the system engineering (particularly in the avionic domain) and in software engineering. Furthermore, conflicts in matching cannot be decided with this technique alone.

10.1.2.11.1.3 Structural matching

In order to overcome the issue raised by name comparison, either assisted by a dictionary or not, more recent attempts have been done to examine not only the element name, but its structure. Authors of the Atlas Model Weaver tool used an algorithm for computing structural similarity based on *metamodel similarity* [Del Fabro, 2007]. A topography of the type of the element being matched is established and compared to the corresponding element in other metamodel. This comparison is done recursively on subcomponents. In this case the answer can only be a matter of probability. Thus, a similarity value is computed, using at the same time name-based matching, name-similarity matching and structural matching. This value allows to offer to the user a similarity proposal, which can be validated or not. The user can in particular discriminate between the conflicting matching. Thus, overlap detection is only semi-automated.

A near approach is followed by [Falleri, 2008], which transforms the source and target metamodels into labelled oriented graphs containing most of the elements of the original metamodel, and then applies a well-known schema matching algorithm called *similarity flooding*. Similarly to the previous technique, it relies on the heuristic stating that similar elements in the meta-model will have similar neighbours. The algorithm computes all elements similarity, floods the values on its neighbours, and then perform a fix point iteration – i.e. iterate until the similarities values converge to stable values for each element of the graph. The correspondences are established between two EMF models in an Eclipse environment.

Both approaches rely on a strong structural similarity between the metamodels and are efficient only if the differences between the metamodels are mostly syntactic. This hypothesis can be true for some viewpoints, but cannot be generalized. In particular, viewpoints can be “orthogonal”, for instance a functional viewpoint of the system against a security viewpoint. Structural matching is likely to be of very little use in this case. Thus, overlap detection must be used with caution in the context of heterogeneous models synchronization.

10.1.2.11.2 Transformation by example

Transformation by example is an alternative method to metamodel overlap detection in order to build semi automatically correspondence links between two metamodels. This approach derives from the Programming By Example approach, although applied to model transformation. The main idea is that the user describes, at model level, actual transformations, and let the transformation engine inferring the correspondences at metamodel level.

An example of such application in the Eclipse environment is presented by the authors of [Varró, 2007], in which the actual transformations are iteratively refined in order to precise or correct the correspondences initially computed by the inductive logic engine. The relation with the issue of transformation specification and design is explicitly cited by authors are one of the main conceptual advantage of the technique (transformation examples being no other thing than use cases). The proposed tool, however, relies on the assumption that merging transformations are forbidden, which seems too restrictive in the case of model synchronization, which often involves partial overlap.

Version	Nature	Date	Page
V01.00	R	2014-04-2830	97 of 117

Another implementation was proposed in [Sun, 2011], which offers a more user-friendly and interactive framework to describe and refine transformations examples (wizards are provided, which allow for instance to define interactively enabling preconditions for transformations). Since the method is more user-based and interactive than the example-based one, the approach is rather called *model transformation by demonstration*. Although the method is intended to endogenous transformations, it can be extended to exogenous model transformations. As opposed to the previous method, it supports partial overlaps, one-to-many and many-to-many transformations.

Both approaches conceptually allow covering more heterogeneous metamodels than similarity-based approaches, independently to the structural similarity. They do not imply relying on dubious external sources such as dictionaries, which is likely to conflict with business or enterprise culture, but rather try to catch the knowledge on the viewpoint semantics. Furthermore, it fits very well in a necessary transformation engineering process (cf. 10.1.2.10).

10.1.2.12 Conclusion

As a conclusion on transformation language, there is no transformation language that displays all desirable properties for model synchronization now. QVT appears to be the least faulty, however, and could be a candidate for one-shot model synchronization. However, as shown in section 10.1.2.10, transformation languages only cover a small part of model transformation engineering, which should also address requirements, high-level and low-level specifications, test and analysis.

This aspect is particularly important when designing synchronizers, which must be maintained during a long period of time, and must synchronize models conforming to evolving metamodels from different stakeholders. Thus, synchronizers are doomed to evolve according to the meta-models they address, and their engineering is a critical aspect of the design framework.

10.1.3 Concurrency and consistency control

Previous chapter has showed that technical aspects of model synchronization were well-studied, although some hard points were still unsolved or were proved intrinsically complex. We claim that operational aspects of model synchronization can solve these hard points and turn heterogeneous model synchronization into an attractive technique for multi-metamodel model-driven architecture.

In our opinion, the main challenge of heterogeneous model synchronization lies in the distribution architecture.

10.1.3.1 Architecture issues

The hard point in the distributed architectures was formalized in the CAP theorem by Gibert and Lynch in [Gibert, 2002], stating that distributed systems have three desirable properties: consistency, availability and tolerance to partition (which means fault tolerance in our context). Over these three properties, only two can be guaranteed at the same time. Thus, any system architecture is a trade-off between these three aspects. The latter sections describe architectures corresponding to the possible combinations of these properties.

10.1.3.1.1 Consistency and availability

Consistency and availability focused systems ensure that data are available at any time and consistent. They, however, do not behave well (i. e. They do not ensure consistency nor availability anymore) when the system is partitioned.

An example of such architecture is a centralized and proxy-based architecture, with heavy clients that possess all the information they need from other models locally. The central node (or bus broker) only transforms model operations into consistent update in other viewpoint and send them to the associated nodes. This architecture is illustrated in Figure 10-21, where $o1$ is the initial model modification in the model 2 and $f1$, $f2$ and $f3$ the functions that define the corresponding operations to be done respectively in models 1, 3 and 4 in order to ensure consistency. These functions are implemented in the *proxy* node, which does not keep data from models.

Version	Nature	Date	Page
V01.00	R	2014-04-2830	98 of 117

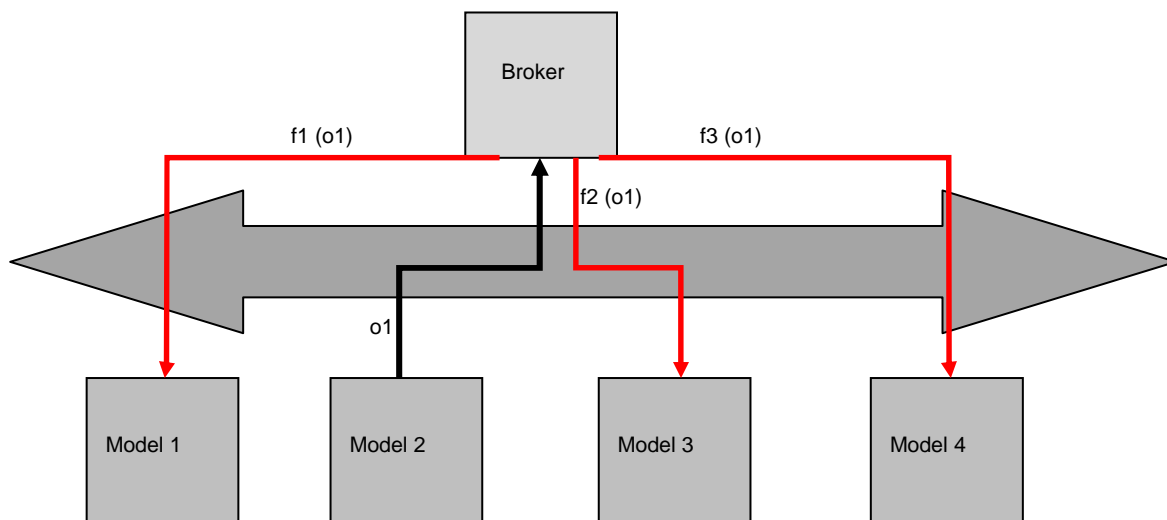


Figure 10-21: Centralized, heavy client, proxy-based architecture

Consistency is ensured by the broker. A synchronous implementation can ensure an even stronger level of consistency: in this implementation, the modification in model 2 is locally validated only after $f1$, $f2$ and $f3$ have been validated. Availability from the user point of view is ensured by the fact that all clients possess all the information necessary from an initial consistent state. Thus it is always possible to edit locally the model (in an asynchronous context). Partition tolerance, in the other hand, is very weak, since all the architecture relies on the broker node – a failure of this last node impedes consistency (but not availability).

10.1.3.1.2 Consistency and partition tolerance

Consistency and fault tolerance ensure consistency over all the models in a resilient architecture. The lack of availability manifests itself by the inability to modify models in some situations. It is implemented in rigid systems which choose security over efficiency. Most existing process in critical industrial projects tend to rely on this paradigm.

An example of such architecture is a centralized proxy-based architecture with light client. This architecture differs from the heavy-client one by the fact that there is no duplication of information between the different models, and that common information is either hosted in the proxy or pointed by it. Thus a model refers to external data and relies on the proxy to get it.

This architecture offers some advantages: strictly local modification can always be done (since no reference is involved, the proxy is not needed), while operations involving external information automatically failed in case of system partition, which ensure consistency of the different models by construction. In fact, this solution is quite similar to the synchronous implementation of the centralized architecture with heavy client. Note that it also shares its flaw: any network problem (which does not necessarily proceed from a partition) will delay or forbid modification operations. This kind of behaviour proved to be very frustrating to the end-user.

10.1.3.1.3 Availability and partition tolerance

The last solution is to modelling consistency for availability and partition tolerance. These systems are reactive to change yet robust, however, at a given time, it is not possible to guarantee consistency of the different models.

An example of this paradigm can be found in pair-to-pair system with heavy clients. In this architecture, each client hosts synchronizer to translate external operations into its own formalism, and then potentially propagates the modification to neighbour if needed. An example of implementation is illustrated in Figure

10-22, where model 1 is modified (operation $o1$) in a way that affect model 2. Model 2 is modified by the operation $f1$, and then updates model 3, which again is modified and propagates the result to model 4.

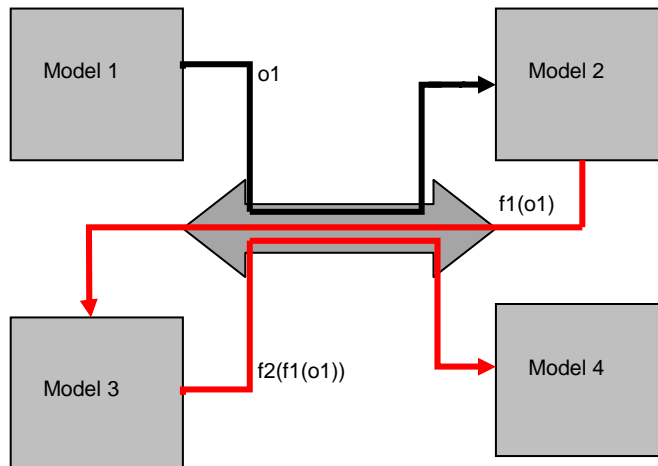


Figure 10-22: Pair-to-pair, heavy client architecture

A variant of this architecture is the fully decentralized architecture, where an operation is sent to all possible interested nodes simultaneously. Although more simple and efficient from a timing point of view, this solution involves more synchronizers (for 4 nodes, 6 synchronizers would be necessary instead of 4 in the pair-to-pair solution). Furthermore, pair-to-pair solution is topology-independent and does not require broadcast capacities.

In pair-to-pair solutions, partition tolerance is very strong since there are multiple paths to update distant nodes. Actually, pair-to-pair paradigm was designed to ensure partition tolerance. Availability is guaranteed by the heavy client. Note that a light client version of pair-to-pair architecture would ensure consistency but would be really inefficient regarding availability since the propagation time of update would produce important delays between command and actual local execution.

10.1.3.1.4 Other desirable properties

Apart from CAP properties, the architecture determines other properties that are important in the context of heterogeneous model synchronization. We saw in chapter 10.1.2 that designing and maintaining synchronizers through transformation languages or any other technique is a difficult task. Thus, it is important to reduce the number of synchronizers running in the system.

The architecture solution is composed of different models that conform to specific meta-models, which address different concerns and are generally designed and maintained by different people, using various, domain-specific or concern-specific platforms not designed to work together. Thus, the system corresponding to the various modellers and tools is not only a distributed system, but a system of systems. As such, one major need is the flexibility or capacity to evolve.

Finally, the visibility management is the capacity of an architecture to guarantee that a given user will only access to the data that he needs in order to perform his task.

10.1.3.1.4.1 Number of synchronizers

In regard to this objective, the architectures presented above are not equal. Centralized solutions with heavy clients are the less efficient, since each client must run a synchronizer for all other clients. Centralized solutions with light clients are more effective, since the translation is done through a pivot model, which factorizes the number of actual transformations to design.

Pair-to-pair solutions can vary greatly in term of efficiency according to the allowed topology. In the example given in Figure 10-22 they are completely serialized. In this case, the number of synchronizers is the smaller

possible ($n-1$ for n meta-models to synchronize). They are, however, really costly in term of transformations actually performed, as they performed unneeded transformations.

In all cases, knowledge of the actual dependencies between the models can reduce the number of synchronizers to design. Such knowledge already exists, since a process, either de facto or de jure, always define the different design stages. In industrial contexts, the process is formalized through various documents. Only actually dependant models should be synchronized.

10.1.3.1.4.2 Flexibility

In the context of a system of modellers and tools, flexibility means mainly the capacity to match in all the system an evolution of a given modeller. The impacting evolution is generally the meta-model evolution. It involves changes in all synchronizers connected to the given modeller.

Regarding flexibility, centralized solutions with light clients is the only efficient solution, since the synchronizers are stored in the proxy and can be changed more easily in a short time.

On the other hand, other architectures impose updates to be performed in all concerned nodes, which is unpractical and is likely to lead to inconsistency between synchronizers and metamodels. Indeed, since the different nodes belong to different stakeholders, to get an agreement on a date and content for the update is a difficult task. Misunderstandings of other side's changes are likely to generate errors in synchronizers.

10.1.3.1.4.3 Visibility management

A last property that is desirable is the visibility or management of each user rights. In the case of centralized architecture with heavy clients, the whole model (or updates) is sent through the network, and only the receiving client will decide which information is needed and which is not. This is a breach of security, since the client can be tweaked by the end-user in order to access unnecessary information.

The same issue appears for pair-to-pair systems, since a client must receive the information it needs, but also the information further pairs would need, which ends up being most, if not all, of the updates.

The centralized solution with light clients offers once again the most interesting results. The whole model or update is always sent to the proxy, which possesses the knowledge of information needed by each client. Thus, it can send only the necessary information, ensuring security of the information.

10.1.3.1.5 Conclusion on distribution issues

Centralized solution with light clients ensures flexibility and security as well as an interesting trade-off regarding the number and difficulty to design synchronizers, while pair-to-pair solutions minimizes the number of synchronizers at the expend of other desirable properties. Centralized solutions with heavy clients performed poorly on all cases.

Regarding CAP properties, most studies have shown that permanent consistency was not needed and could even be harmful as it prevents creativity and innovation in design, as shown in an industrial context by the authors of [Nissen, 1996]. On the other hand, partition tolerance is more and more needed with the rise of nomad computing which leads to frequent disconnections. Availability is also an important feature, with users being used to highly responsive systems. As a conclusion, we would rather choose a solution ensuring availability and partition tolerance.

An interesting solution would be to associate both solutions: a backbone of pair-to-pair organized brokers sharing instances of the current model formalized in a pivot model, with buses associating clusters of light clients. The information actually stored at each level should be the subject of an in-depth study, depending largely on the process. Connexions between the brokers should also be process-dependent. An example of such architecture is given in Figure 10-23. In this example, availability and partition tolerance is ensured between the brokers, since they are organized in pair-to-pair or similar distributed architecture. Since there is no actual model transformation between two brokers thanks to the pivot model, this solution involves no extra cost regarding the number of synchronizer to design and maintain. Flexibility can be kept low if the

Version	Nature	Date	Page
V01.00	R	2014-04-2830	101 of 117

clusters are organized by data dependencies. Finally, visibility management is ensured at client level since all operation sent to a client passes through a broker which performs filtering.

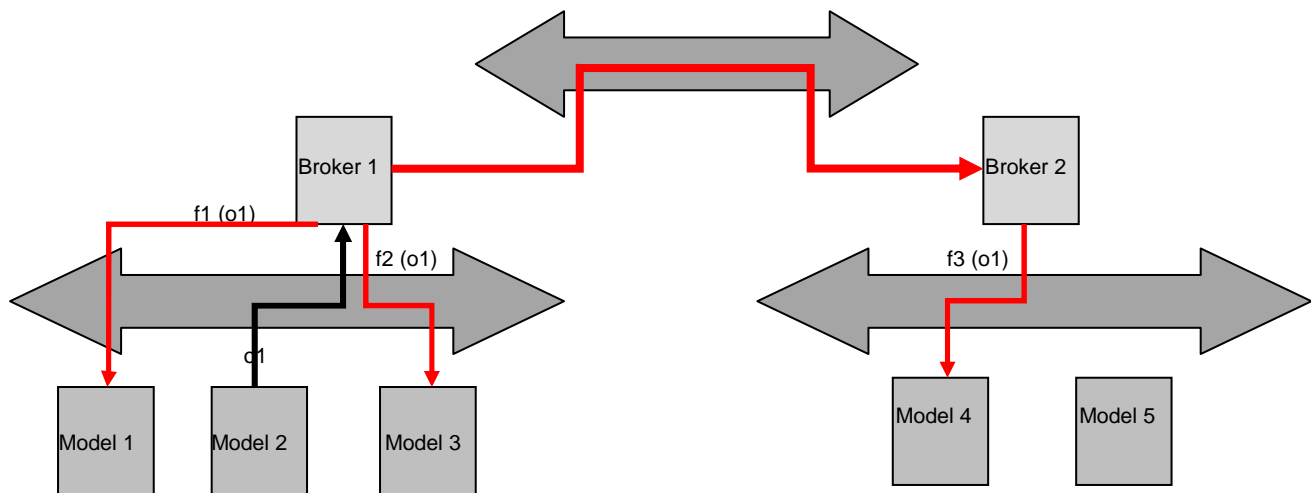


Figure 10-23: Multi-cluster architecture

10.1.3.2 Consistency issues

Multiple solutions have been found to ensure consistency in a context of distributed concurrent edition, mainly in the domain of groupware. This domain was found by [Ellis, 1989] to have the following properties:

- High interactivity – which implies short response time ;
- Real-time ;
- Distribution – and the need to support multiple nodes and even multiple networks – barring some options like the use of synchronized networks ;
- Volatility – meaning that users can connect and disconnect at any time ;
- Ad hoc – the impossibility to tell which resource the users will access a priori ;
- Focus – high degree of access conflicts on the same resource during a session.

Apart for the ad hoc criterium, which should be enforced by the process and specific rights and visibility, we claim that these criteria are relevant for distributed modelling. Thus, we study in this section some techniques used by the groupware community to ensure consistency of the data across the distributed system.

10.1.3.2.1 Pessimistic approach

Pessimistic approach is the simplest form of edition in a concurrent context: it consists in forbidding any concurrency in edition. Any client can access a distributed resource in read mode, but only one is allowed to edit it at the same time. This technique is used in Microsoft Word while accessing files on a network drive.

Such policy usefulness depends on the granularity of the resource and the kind of usage. Fine-granularity of data can be obtained by dividing them into multiple files, or with proper API in the distributed application. In both cases, a risk of interlocking arises. This approach does not allow concurrent global edition by definition, and thus is unfit to model edition. Furthermore the pessimistic approach is dangerous in the case of faulty connection or nodes, since a file may stay locked if its current editor node becomes unreachable.

10.1.3.2.2 Operational transformation

Operation transformation is a well-known technique in textual collaborative edition domain consisting in executing local modification immediately after user request, in order to obtain optimal local responsiveness. Then, the corresponding modification is transferred to remote clients. A recent use of this technique can be included in Google Wave ([Mah, 2010]).

Whenever a local modification is done concurrently with a distant update message reception, the update causal vector is checked against the local state of the client. In a concurrent text edition context, the problem addressed is the following: Let us suppose user A types the string “hello!” in his local client.

➤ Hello!

The string appears locally and is sent to distant client. Let us suppose client B receives the update, and adds the “world” string (space included) before the exclamation mark (i.e. after fifth character). Typically, a primitive consisting in *insertAfter*(4, “ world”) would be generated. The following result will appear locally to B:

➤ Hello world!

However, if A concurrently tried to delete the exclamation mark (*deleteChar* (5)), a naïve management of concurrency leads to the following result for when B’s update is executed before A’s.

➤ Helloworld!

In which the space at position 5 disappeared, which does not preserve neither user intent: deleting the exclamation mark (A intent) and adding the “world” string including the space.

Furthermore, the result could be different in B’s client, leading to inconsistency between the two texts.

Operation transformation consists in this situation in transforming the later update according to earlier updates which are not known at later request emission time. Each message is sent altogether with a causal vector indicating which remote updates it is taking account of. At reception by a remote client, the vector is checked against the current local value. If the local value is greater than the vector corresponding position, the string received is modified according to pending local updates. In this particular time, the problem can be fixed by using offsets rather than positions in the string.

If the reconciliation policies (or *operation composers*) are consistent across the network, the string will be consistent at edition time, which is a satisfactory condition in most cases.

Operational transformation allows to works on unstable networks, since it can infer the actual local transformation according to the respective states of emitters and receivers. Large updates without synchronization are likely to be lost, however since the elements referenced might not be present anymore.

More importantly, the composition of graphical updates is known as a difficult task, and no academic or industrial results have been produced to our knowledge regarding the composition of graphical rich environments.

10.1.3.2.3 Three-way merges

Three-way merge is an approach used in repositories (for instance in Subversion). It is a centralized approach which uses the following technique:

1. The client sends the content of a document to a node whose status is server ;
2. The server performs a merge of the data from server, user’s client changes and changes from other users (hence 3-ways) ;
3. The server sends a new copy to the client.

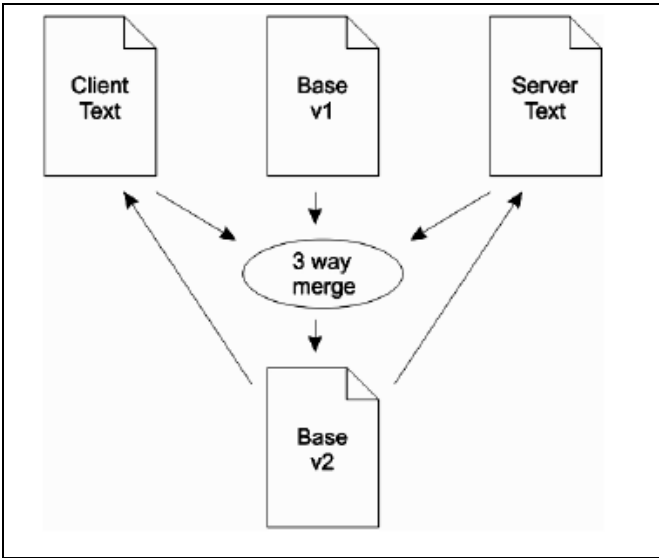


Figure 10-24: Three-way merge

Figure 10-24, extracted from [Fraser, 2009], illustrates the three-way merge. Apart for being centralized, the specificity of three-way merge compared to operational transformation is that the “base” is actually the base common ancestor between a central repository and the edited version. In the case of Subversion, if a data is updated concurrently by two users, the three involved version will be the one of the two users and the common ancestor on the base.

Generally, actual implementations are capable of resolving some simple conflicts when they do not overlap in the same paragraph. Conceptually it is a mix from the two previous approaches, which offers a good trade-off between concurrency and feasibility by adding some manual reconciliation work for users. However, this method put a heavy load on the network since the whole modified files are sent to the server. In a dynamic context, this approach is not realistic. Note that ClearCase’s *Dynamic Views* are actually able to perform dynamic merges. However, which merging strategy is used is currently unclear for the authors.

10.1.3.2.4 Differential synchronization

Differential synchronization is a new approach proposed by [Fraser, 2009]. It aims mainly to address the network issue of the three-merge approach.

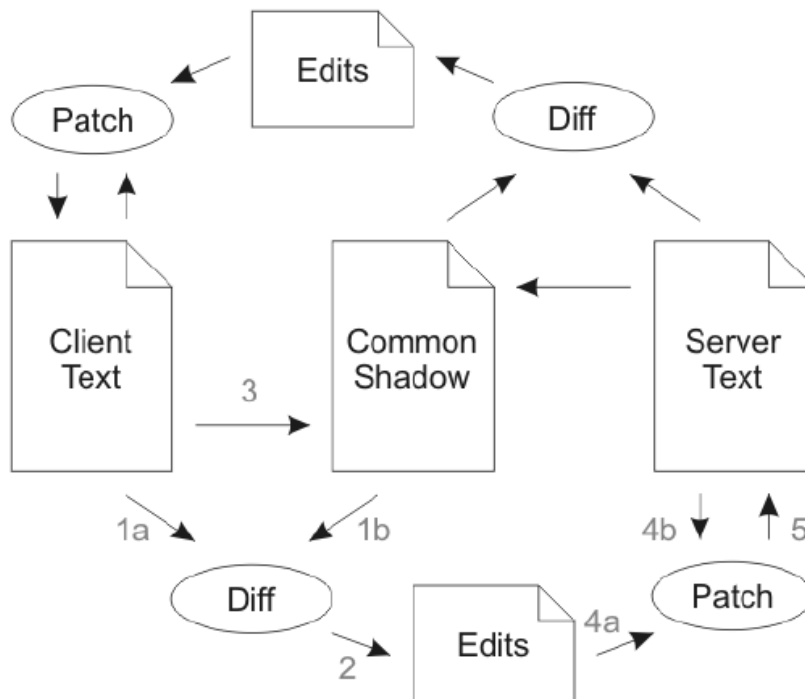


Figure 10-25 : Differential Synchronization workflow

Figure 10-24, illustrates the differential synchronization workflow. *Client text* and *server text* are actually files in the same (client) node. *Server text* is actually an image of the server stat, and can be edited concurrently at any time by a dedicated thread. Initially, *Common shadow*, *server text* and *client text* are assumed equal. The differential synchronization follows these steps:

1. Client text is diffed against common shadow ;
2. A list of edits is created, which contains the client changes comparatively of the last version received from the server ;
3. The client text overwrite the common shadow ;
4. The edits are applied to the server on a best-effort basis ;
5. The server text is updated according to the patch ;
6. The process is then symmetrical in the server-to-client direction.

Note that actions 1, 2 and 3 are atomic: the client text is in read-only mode before overwriting the common shadow is complete. Actions 4 and 5 are also atomic.

The system is dynamic: the cycle is not triggered by an edition of either user or server, but is periodical. The setting of the period is quite important, and must be a trade-off between system load and system responsiveness.

10.1.3.3 Inconsistencies in heterogeneous models

10.1.3.3.1 Heterogeneous models consistency

The increasing adoption of Model Driven Engineering by the industry leads to the simultaneous existence of multiple models which, although conforming to different formalisms, represent different concerns on the same physical objects, including specific analysis or simulation tooled formalisms. These formalisms may also have different granularity and level of abstraction.

The heterogeneous hypothesis – which is mandatory in real-life system where different stakeholders have defined their own tools and formalisms for their specific concerns – implies that there is no formal link

between the elements of the different models. Instead, there is a *semantic relation* which is often partially understood by their respective designers and much less by the end-user of the modelling tool.

Hence inconsistencies are likely to arise soon from the different models, except if there is a rigid process ensuring consistency. This last hypothesis, mainly implemented through sequential and blocking stages of modelling, although widely adopted in the industry, is clearly suboptimal since it forbids any parallel design. From our point of view, it is bound to disappear in all but the most critical part of the design stage.

Indeed, at some point of the design process, these models must have some level of consistency. Ensuring consistency has proved to be a difficult task. Consistency, however, is not needed at any time. For practical reasons (e.g. bandwidth, modeller response time, etc.), it may be non-optimal. Depending on the nature of the models and their semantic relations, temporary inconsistent states may be unavoidable during design.

As demonstrated in an industrial context by [Nissen, 1996], inconsistencies are not errors and are not even undesirables: actually, inconsistencies are often caused by a misunderstanding between the different designers on the semantical relation existing between the different models or deep disagreement over the system architecture. On both cases, such issue must be investigated deeply, either for the sake of the product or the process. Hence, inconsistency detection is not merely a supplementary step resulting from the introduction of a new technology (concurrent heterogeneous modeling), but it is also a way to acquire a better understanding of the system and the stakeholders viewpoint of it. These benefits will only appear if some degree of laxity is let to the designers.

Consistency control is closely tied to model synchronization, since both involve ensuring to all users that the data they need are up-to-date. It is however an orthogonal field to model synchronization, since it focuses on implementation level, while model synchronization focuses on specification level. In particular, it does not specify how the heterogeneous models are mapped. Studied techniques were all tested on somewhat trivial (partially bijective) mappings. Another difference is that consistency is only defined between pairs of models (or pairs of model sets), while model synchronization considers more than two protagonists. Consistency control is however used since the late eighties and, as such, offers mature techniques which applicability to model synchronization should be seriously considered.

10.1.3.3.1 Illustration of inconsistencies issues

Let's take the use case of the design of a distributed application. In this example, the physical network on which the application is deployed is designed in parallel with the application. Both are represented explicitly with a model. Figure 10-26 illustrates a model representing a physical network of computers. Another model of the same network is illustrated in Figure 10-27. It represents the services exchanged between the computers. The latter model describes a synchronization request send by N1, which triggers a synchronization of all computers except N2 on N3 clock.

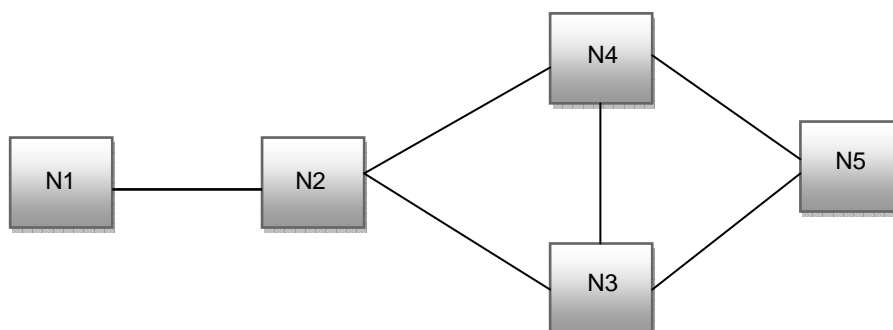


Figure 10-26: Physical network model

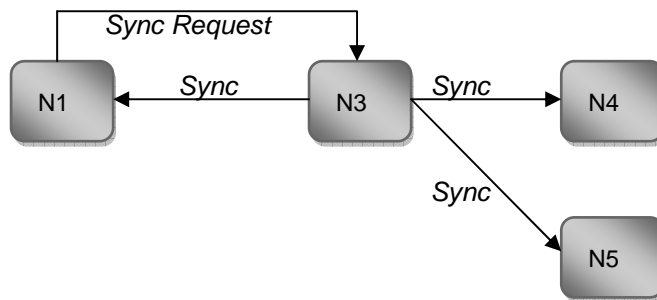


Figure 10-27: High-level services model

Although both models represent the same real-world network, they differ significantly:

- N2 is only a router, thus does not provide high-level services ;
- The connection between N2 and N4 is not used by any services ;
- Although there is no direct connection between N1 and N3, services are exchanged between these two nodes.

The last point is an implication of our definition of services: a service can be provided to a computer if it is directly or indirectly connected in the physical network model. Low-level services (including network-level exchanges) would indeed be more restrictive. This rule is a *consistency rule*: the two models are consistent if and only if this rule is verified.

Thus our models differ at the same time in their concerns (physical connections versus actual exchange) and in their level of abstraction (low-level versus high level). Let's make a simple change in the first model (Figure 10-28).

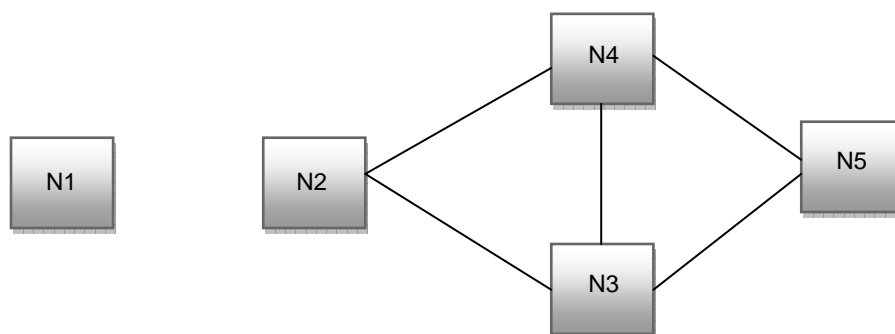


Figure 10-28: Modified physical network model

By simply deleting the physical link between nodes N1 and N2, the high-level services network is made inconsistent with the physical network model, since nodes N1 and N3 are no more connected, either directly or indirectly. Harmonizing the two models can be done by deleting faulty links in the high-level services model, as shown in Figure 10-29.



Figure 10-29: Harmonized high-level services model

The consistency rule between the two models is no more violated. Is it to be noted, however, that synchronization is done without synchronization request. Although this is not an inter-model inconsistency, it is probably an error. Thus inter-model inconsistency must be aware of intra-model consistency rules, since correction of inter-model inconsistencies can lead to violating them.

10.1.3.3.2 Inconsistencies definition

There is basically two forms of inconsistencies:

- Intra-model inconsistencies (e.g. well-formedness) ;
- Inter-model inconsistencies.

We focus only on inter-model inconsistencies, since intra-model inconsistencies are well-covered by the industry practice.

In previous section we informally defined an inter-model inconsistency in terms of the violation of a *semantical relation* between two elements from different models. Authors of [Spanoudakis, 2001] propose a more formal definition.

The authors introduce the notion of *interpretation*, which describes the relation between the actual individuals from a hypothetical domain model and the elements present in a stakeholder's viewpoint. *Overlap* between different interpretations is defined if and only if elements from both interpretations are related to the same domain individuals. Since the domain is often practically impossible to bind formally, an alternative is representing overlaps as relations between the E_i and E_j , the sets of elements of two software models, the set of the agents who can identify them (A) and the set of overlap kinds (O_T).

$$OV \subset E_i \times E_j \times A \times O_T$$

Where the set of overlap kinds includes *no* (null overlap), *to* (total overlap), *po* (partial overlap) and *io* (inclusive overlap – e_i overlaps inclusively with e_j). It is to be noted that the meaning of an interpretation is not covered by this definition: only the relationship existence is displayed.

With those definitions, an inconsistency can be defined. Spanoudakis et al. propose the following definition:

Definition. Assume a set of *software models* $S_1 \dots S_n$, the sets of *overlap relations* between them $O_a(S_i, S_j)$ ($i=1, \dots, n$ and $j=1, \dots, n$), a *domain theory* D , and a *consistency rule* CR . S_1, \dots, S_n will be said to be inconsistent with CR given the overlaps between them as expressed by the sets $O_a(S_i, S_j)$ and the domain theory D if it can be shown that the rule CR is not satisfied by the models.

Consistency rules may be:

- A *well-formedness rule*, an intra-model specific rule that defines the conformance to the model formalism.
- A *description identity rule*, specifying identical identifiers between totally overlapping elements from different metamodels.
- An *application domain rule*, specifying relations between individuals from the domain addressed by the models elements.
- A *development compatibility rule*, probably the most important kind of rules in our perspective, which specifies consistency between the different models elements.
- A *development process compliance rule*, which describes the conformance of models to standards and norms.

In the use case illustrated in previous section, the rule “A service can be provided to a computer if it is directly or indirectly connected in the physical network model.” is a development compatibility rule, while the rule “synchronization must be preceded by synchronization request” is an application domain rule.

In respect to development compatibility rules, the authors of [Van Der Straeten, 2003] propose a classification of inconsistencies, between structural and behavioural inconsistencies: structural inconsistencies arise when the structure of a model is incomplete, incompatible or inconsistent with respect to existing behaviour. Another dimension of inconsistency, is the type of the affected model, i.e. its level of abstraction (model, meta-model, meta-meta-model), as illustrated in Figure 10-30. This last dimension only

Version	Nature	Date	Page
V01.00	R	2014-04-2830	108 of 117

has a meaning in the case of EMF modelling – the meta-model is not always modelled in actual industry, which often only design.

	Behavioural	Structural
Model-Model		dangling (type) reference inherited association conflict
Model-Instance	incompatible definition	instance definition missing
Instance-Instance	invocable behaviour conflict observable behaviour conflict incompatible behaviour conflict	disconnected model

Figure 10-30: Development compatibility inconsistencies

10.1.3.4 Detecting inconsistencies

10.1.3.4.1 Inconsistencies detection

Techniques, methods and tools to detect inconsistencies have been developed since the late eighties. They are based on two different approaches:

- Logic-based approaches
- Pattern matching approaches

10.1.3.4.1.1 Logic-based approach

The logic-based approach consists in modelling formally the system and then using formal inference techniques in order to derive inconsistencies. An example of formal modelling languages is the B notation from B-Method¹⁰, derived from first-order logic. It implies a reformulation of the definition of inconsistency, by introducing:

- The notion of a set of inference rules:
- The notion of consequence of a given operation (according to a set of reference rules).

Most of those techniques operationalize the consequence relationship using theorem proving approaches based on standard inference of first-order logic such as resolution, conjunct and negation elimination.

The satisfiability problem is undecidable in first-order logic, thus authors of [Van Der Straeten, 2003] proposed to use a decidable fragment of first-order logic aiming at describing formalisms: Description Logics. Description Logics is a two-variable offering a classification task based on the *subconcept-superconcept* relationship. This classification task is decidable and complete. This formalism is used to describe the consistency problem at metamodel level on software models (UML), which implies to rewrite them. Then, an inference engine can reason on the instances and find out inconsistencies between them.

All logic-based approaches assume that the models can be specified formally, which is generally not the case, because it requires a strong expertise from the designer. It is a difficult problem to define a language that is both formally well-grounded and aligned with business concerns and habits.

10.1.3.4.1.2 Pattern matching approaches

Pattern matching is the most usual way to detect inconsistencies between models. In a traceless environment, this approach consists in matching the left-hand expression in input models and then matching the corresponding right-hand expression in output models. Failure in matching the right-hand implies an inconsistency between source and destination models.

¹⁰ <http://www.methode-b.com/en/>

Authors of [Tsiolakis, 2000] use this approach in order to detect inconsistencies between UML diagrams (class diagram and sequence diagram). They use attributed graph grammars. In this use case, each class of the class diagram is transformed into a node in the type graph. Relations between classes (inheritance, generalization, composition, etc.) are modeled as arcs, whose attributes are properties of association like name, role or visibility. Finally, association multiplicity is represented by constraints which belong to the rules left-hand (precondition). The sequence diagram is translated into a start graph and rules (i. e. a graph grammar). Here, rules are used to model message production. Pattern matching is used to ensure that all productive rules from the sequence diagram are grounded in the graph extracted from class diagram.

Note that the expressivity of general graph grammar is known to be too weak to define some inconsistencies. Thus pattern-matching based inconsistency may need to use more expressive formalisms such as programmed graph grammars. Furthermore, graph grammars are quite inefficient on large models.

10.1.3.4.2 Incremental inconsistencies detection

The general idea of incremental checker is to perform *operation-based* checking that is, instead of pointing out the problematic elements in the models, to identify the problematic actions which have (supposedly) caused the problem.

Incremental inconsistency detection is only possible if it is possible to describe the difference between two models in terms of modification operations. Typically, the approach starts from known-consistent models, and perform inconsistency detection whenever one of the models is modified, taking the modification as a parameter. This, of course, implies the existence of an API on the modelling tool which provides notification services. This is the case in most modern modeller such as Eclipse EMF.

Since there is a very limited set of atomic possible modification on a model (typically only four of them: *create*, *delete*, *setProperty* and *setReference*), it leads to a small number of rules to be triggered, with typically very small left-hand. On the other hand, the right hand which describes the operation expected in the models to be updated tends to be very complex. It is to be noted that the rules are bidirectional only if the models are partially bijective.

Authors of [Blanc, 2008] use this approach by writing logic formulae over the sequence of both source and destination models construction operations. A Prolog inference engine is then used to validate the predicates. In order to be more expressive, authors do not restricted themselves on first-order logic, but used Prolog facilities such as manipulation primitive types (lists etc.). Since rules describe the operation history on both sides (source model and destination model), it is a very efficient solution in term of bandwidth and computation. Furthermore, it natively allows keeping a model history with zero cost.

Further works of the same authors proposed techniques of rule reduction and scope reduction according to the operation performed. These extensions exploit the known semantic of the operation in incremental checker (as opposed to checkers working on black-box models) in order to optimize the performances and scalability of the detection. Usage of cache is also discussed.

Is it to be noted that the initial hypothesis of consistent models can be bypassed, supposing that one has the capacity to decompose a model (or a model delta) into a canonical set of atomic operations (not necessarily the one that was actually used to build the model). For instance, if a component property is updated multiple times between two inconsistencies detection, only the last update would be present in the canonical decomposition of modification operations. This can apply as well in regular incremental inconsistency detection, since it allows minimizing the number of rules to be triggered.

10.1.3.5 Model harmonization

Once inconsistencies detection has been performed, a way to manage them must be found. This section presents briefly some techniques in order to harmonize the models for the different techniques of inconsistency detection.

Version	Nature	Date	Page
V01.00	R	2014-04-2830	110 of 117

10.1.3.5.1 Non-incremental approaches

Non-incremental matcher makes quite difficult to fix inconsistencies. Graph grammar typically lacks the expressiveness to describe complex real-life model operations, which are likely to need iterations, indirection, conditional jumping, etc. This is why most transformation languages also offer an action language (eg. ATL, QVN) to express transformation rules to be applied in each case.

The transformation rules are however typically not independent from one to another: two rules can be mutually exclusive or one rule can trigger another one, because it makes previously non-matched rules now matching. Thus, it is necessary to compute a sequence of rules which actually leads to the proper model without adding more inconsistencies, if necessary by adding new transformations. This task is named *transformation dependency analysis*.

Such work was done in [Mens, 2006]. A distinction is done between parallel independence (absence of mutual exclusion) and sequential independence (absence of causal dependencies). They use critical pair analysis techniques in order to compute all possible mutual exclusions and sequential dependencies for a given set of transformation rules by pairwise comparison. It is to be emphasized, however, that this method may lead to false positive conflict detection.

10.1.3.5.2 Incremental approaches

As opposed to non-incremental approaches, incremental approach accesses to the models history. Hence, they can distinguish between conflicting models and model not up-to-date. In the last case, the harmonization is straightforward, since the left-hand of the transformation defines a set of operations that simply has to be done to enforce the rule.

A two-step approach for model harmonization is proposed by authors of [da Silva, 2010]. It consists in 1/ enumerating possible way of changing an inconsistent action and 2/ generating a repair plan for the sequence model.

The first stage uses generator function, describing operations to be performed in destination model with respect to the original operations in source model. These right-hand operations are written manually, since some choices have to be done (new object naming, default value setting, etc.).

The second stage uses a specific exploration strategy in order to explore the impact of the different sequences of possible actions (*tree of possible repair plans*). The exploration ceases when a consistent model is found or no more action is possible. In the latter case, it backtracks to the previous exploration choice done and explores another branch of the tree. The cycle iterate until a consistent model is finally found or when a maximum number of allowed iterations is reached. In order to make the search more efficient, a heuristic is proposed to sort the operations from the most recent to the most ancient, with the idea that more recent modifications are more likely to break consistency than the old ones. As opposed to non-incremental approaches, it can find a satisfying solution if one exists in the allowed scope of iterations.

10.2 Annex II: Co-Evolution Specification

10.2.1 Problem

Applying Model-Based Engineering in industrial processes often results in the production of a set of models which are technically disjoint but conceptually connected and evolve in parallel. This situation typically happens:

- when the workflow involves several modelling tools which are dedicated to different tasks, for example a system modelling tool and tools for model-based analysis or simulation;
- when the system to design is complex enough to require the construction of different models dedicated to different sub-systems or levels of abstraction;
- when organisational boundaries between system integrators and sub-contractors bring confidentiality constraints, so each company has to work on its own private model and shares only the subset related to integration interfaces;
- when a metamodel integrates many different concerns: reuse, traceability, suitability to different modelling methodologies, etc., and an abstraction of the models focussed on a single concern is required at some point in the modelling process.

Rather than implementing from scratch a bridge between models every time those needs arise, it would be most beneficial to rely on a technology that already supports the parallel evolution of models, or “co-evolution”, as much as possible.

10.2.2 Positioning Overview

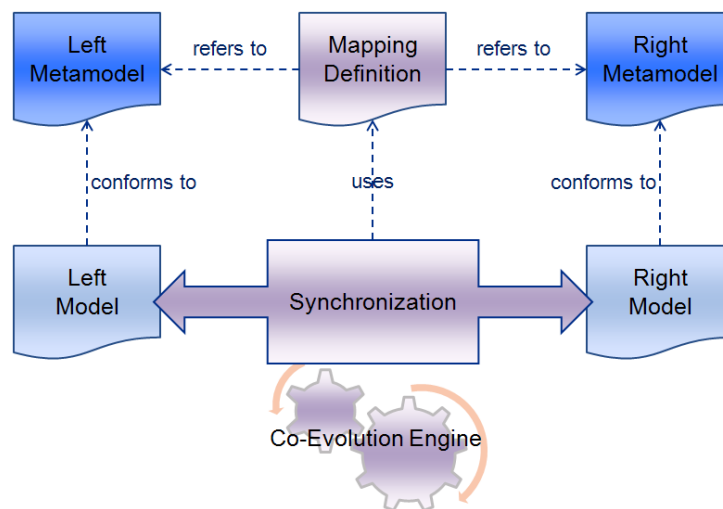
A technology for model co-evolution would ideally be such that:

- it is based on a mapping from source metamodel(s) to target metamodel(s), which is defined at a high level of abstraction in order to be easy to write and maintain;
- it supports model update so that models that have evolved synchronise: neither model is fully re-generated;
- it supports semi-automatic synchronisation in the cases when human intervention cannot be avoided;
- it supports, when relevant and as much as possible, bidirectionality: the same (subset of) mapping can be executed in either direction.

There exist many languages that are dedicated to the manipulation of models, even though, as shown in the State-of-the-art Annex of this document, they do not fully cover the need for model co-evolution. Consequently, rather than creating a new language, we primarily intend to define a framework that is able to integrate existing languages and technologies. This is especially true for model query technologies such as OCL, Acceleo, Model Query or EMF-IncQuery, which can be useful to express co-evolution mappings. An execution platform dedicated to this framework is required in order to execute the mappings (figure below).

In accordance with these high-level positioning principles, the next section provides a precise set of requirements on the co-evolution technology.

Version	Nature	Date	Page
V01.00	R	2014-04-2830	112 of 117



Execution principle for the model co-evolution technology

10.2.3 Requirements

10.2.3.1 Main features

This subsection is concerned with the main, coarse-grained features of the co-evolution technology.

Ref	Title	Description
1	<i>Transformation</i>	The technology shall be able to generate a Target model from a Source model.

Ref	Title	Description
2	<i>Iterativity</i>	The technology shall be able to re-generate a Target model from a Source model after the Target model and/or the Source model have evolved in such a way that the Target model is updated. How update occurs shall be configurable.

Ref	Title	Description
3	<i>Interactivity</i>	When iterativity is involved, the technology shall allow the user to make choices about the way the Target model is updated. This user interaction shall occur via a GUI.

Ref	Title	Description
4	<i>Bidirectionality</i>	When appropriate and possible, a subset of the (Source, Target) mapping shall be reusable for defining an opposite transformation where the Source and Target roles are swapped.

10.2.3.2 Level of abstraction

The co-evolution technology proposes concepts for defining mappings. This subsection is about the level of abstraction of these concepts, especially compared to traditional “low-level” programming based on the de-facto standard modelling technologies (Java/EMF). The higher the level of abstraction is, the simpler and more maintainable mappings are.

Ref	Title	Description
5	<i>Declarative nature</i>	The concepts proposed by the technology shall manifest a relation between the contents (elements, reference values, attribute values) of a Source model and a Target model. This relation, or mapping, shall be executable.

Ref	Title	Description
6	<i>No intermediate state</i>	Within the mapping relation, the contents of the Target model shall be defined as a pure function of the contents of the Source model. In other words, no intermediate state of the target model shall be visible to the designer of the mapping.

Ref	Title	Description
7	<i>No order</i>	The order in which the transformation steps are executed shall be transparent to the person in charge of specifying the mapping and shall have no impact on the final result.

Ref	Title	Description
8	<i>Accessibility</i>	<p>The core concepts provided by the technology for defining mappings shall be few, simple to grasp and orthogonal.</p> <p>This requirement can be partly tested by comparing the technology with existing transformation technologies/languages and gathering feedback from software engineers.</p>

Ref	Title	Description
9	<i>Determinism</i>	Mapping execution shall always be deterministic, except for the interactive part of the execution.

Ref	Title	Description
10	<i>Iterative vs. non-iterative</i>	There shall be no distinction between Target creation and Target update: only Target description shall be specified in a mapping. The way update occurs shall be specified separately from that description, via a configuration. This high level of abstraction has the advantage of providing consistency guarantees.

Ref	Title	Description
11	<i>Configuration of iterativity</i>	Iterativity configurations shall be specified at a high level of abstraction, that is, in terms of what is a significant change and which changes are accepted or rejected.

Ref	Title	Description
12	<i>Consistency of iterativity</i>	Iterative executions shall always result in consistent Target models, whatever the way they are configured with respect to change reconciliation.

10.2.3.3 Expressiveness

Similarly to the level of abstraction, expressiveness is a fundamental property of the set of concepts provided by the co-evolution technology. It determines whether mappings of an arbitrary complexity can be defined, or whether there are cases that cannot be entirely dealt with using the sole co-evolution technology. A high expressiveness can sometimes be difficult to combine with a high level of abstraction.

Ref	Title	Description
13	<i>Data manipulation</i>	It shall be possible to define the Mapping using a mature, rich, and extendible standard library for data manipulation, such as that of a programming language.

Ref	Title	Description
14	<i>Mapping expressiveness</i>	<p>The technology shall make it possible to define arbitrarily complex mappings while preserving the iterativity and interactivity capabilities.</p> <p>Examples are needed to validate this requirement. An example is: Given a Source model that solely contains a Component A with M States and a Component B with N States, M and N being unknown, there must be in the Target model a single corresponding Component AB with MxN States, where each State of AB corresponds to a couple made of a State from A and a State from B.</p>

Ref	Title	Description
15	<i>Flexibility of iterativity</i>	Iterative executions shall be configurable so that every possible way of reconciling changes from the Source and the Target models shall be expressible.

10.2.3.4 Technological concerns

Beyond its functional qualities, the relevance of the co-evolution technology also depends upon a few essential technical characteristics.

Ref	Title	Description
16	<i>Scalability</i>	The scalability properties of the technology shall be as close as possible to those of lower-level, de-facto standard technologies (Java/EMF). In other words; the overhead introduced by the higher level of abstraction of the technology shall be well understood and justified.

Ref	Title	Description
17	<i>Integration</i>	The technology shall integrate into and interoperate with the de-facto standard modelling technologies: Java and EMF. In particular, configuring and executing a synchronisation shall be feasible solely via Java class instantiations and method calls.

Ref	Title	Description
18	<i>Development environment</i>	The technology shall offer to the designer of mappings a high-level development environment providing classical features such as an editor with syntax highlighting and completion, and a debugger.

Ref	Title	Description
19	<i>Modularity and reuse</i>	The technology shall provide reusable building blocks for defining mappings, so that mappings can be defined from other mappings or parts of them.

Ref	Title	Description
20	<i>Mapping as model</i>	The technology shall support the specification of a mapping as a model, hence making it easy to manipulate mappings and making it possible to define higher-level mappings, i.e., mappings that generate mappings.

10.2.3.5 Versatility

Versatility is expected to be a fundamental quality of the co-evolution technology. It determines to what extent the technology can be used in a wide range of situations and contexts, from usage scenarios to technological configurations.

Ref	Title	Description
21	<i>Metamodels</i>	The technology shall support the definition of mappings from any metamodel to any other metamodel, without technological restrictions on how metamodels are defined.

Ref	Title	Description
22	<i>Persistence</i>	The technology shall be applicable to models which are persistent or not and, if they are, to models which are persisted using any kind of persistence technology such as XML files, binary files or databases.

Ref	Title	Description
23	<i>Data scopes</i>	The technology shall make it possible to arbitrarily define scopes, within models, in which the transformation/synchronization occurs.

Ref	Title	Description
24	<i>Trace system</i>	The technology shall accept different techniques and technologies for handling transformation traces. Transformation traces define how model elements match in order to realize iterativity.

Ref	Title	Description
25	<i>Query technologies</i>	The technology shall allow mappings to be specified using third-party query technologies.

Ref	Title	Description
26	<i>Extendible GUI</i>	The technology shall make it possible to modify or replace the GUI which it proposes by default for interactive executions.

Ref	Title	Description
27	<i>Parameterization</i>	The technology shall make it possible to define parameterized mappings and pass actual parameter values at execution time.

Ref	Title	Description
28	<i>Data/model technology adherence</i>	<p>The technology shall make a clear distinction between concepts that are data technology-agnostic and concepts that depend upon the de-facto standard modelling technology EMF.</p> <p>This is important for conceptual soundness and for anticipating a possible extension of the technology to non-EMF artefacts, and it has to be taken care of from the beginning of the design phase in order to prevent expensive refactoring.</p>