Deliverable D 2.1
Specification of formal development demonstrator

Project acronym:	4SECURail
Starting date:	01/12/2020
Duration (in months):	24
Call (part) identifier:	H2020-S2RJU-2019 / S2R-OC-IP2-01-2019
Grant agreement no:	881775
Due date of deliverable:	Month 06 (May 2020)
Actual submission date:	02/06/2020
Responsible/Author:	Franco Mazzanti - CNR
Dissemination level:	PU
Status:	Issued

Reviewed: YES

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 881775.

Document history		
Revision	*Date*	*Description*
0.1	10/05/2020	First Draft under review
0.2	22/05/2020	Second Draft under review
1.0	27/05/2020	Issued
2.0	15/10/2020	New submission changing disclaimers

Report contributors		
Name	Beneficiary Short Name	Details of contribution
Franco Mazzanti Davide Basile Alessandro Fantechi Stefania Gnesi Alessio Ferrari	CNR	Overall contribution to deliverable structure and context.
Andrea Piattino Laura Masullo Daniele Trentini	SIRTI	Overall contribution to deliverable structure and context.
Carlo Vaghi	FIT	Reviewer
Jeronimo Padilla	ARD	Reviewer

Disclaimer

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The content of this document reflects only the author's view – the Joint Undertaking is not responsible for any use that may be made of the information it contains. The users use the information at their sole risk and liability.

The content of this document does not reflect the official opinion of the Shift2Rail Joint Undertaking (S2R JU). Responsibility for the information and views expressed in the deliverable lies entirely with the author(s).

Table of Contents

1 Executive Summary

The overall goal of the Workstream 1 "*Demonstrator Development for the use of Formal Methods in Railway Environment*", spreading on the activities of Tasks 2.1, 2.2, 2.3 2.4 of the 4SecuRail project is:

- the definition of a "formal methods demonstrator process" (shortly *Demonstrator*) for the rigorous construction and analysis of system specifications (from the point of view of infrastructure managers).
- the application of the Demonstrator process to a railway signalling system case study,
- with the goal of performing a cost benefits analysis and the evaluation of the required learning curve for the application of this Demonstrator process.

This Deliverable "*Specification of formal development demonstrator*", describing the result of the first part of Task 2.1, presents the overall structure of the Demonstrator process and illustrates the selected choices for its architecture, both in terms of methodologies and tools.

The specified formal development demonstrator will be experimented with its application to a simple initial case study in the second part of Task 2.1.

The experience gained with this initial experimentation will result in the consolidation of the definition of the Demonstrator process prototype (reported in the Deliverable D2.2 of Task 2.1 "Formal development demonstrator prototype - 1st release"). The consolidated process will then be applied in Task 2.3 to the complete case study defined in Task 2.2 and that activity will provide the reference for the costs-benefits analysis of Task 2.4.

Before the presentation of the overall structure and architecture of the planned formal methods demonstrator process, three important issues deserve a specific analysis and discussion:

- the clarification of the usefulness of formal methods from the point of view of the Infrastructure Managers,
- the relation between our demonstrator and other relevant projects like Eulynx and X2Rail2,
- the role that the semi-formal SysML notation should play within our formal methods demonstrator process.

The choice of which specific MBSE framework will be used for the semi-formal modelling of the system under design has been deferred to a later stage, when more hands-on experience has been gained with the various possibilities. Instead, the choice of which verification technique will be used has converged to the model checking approach as supported by the Even-B methodology and the ProB framework.

2 Abbreviations and acronyms

Abbreviation / Acronyms	Description
ATP	Automatic Train Protection
ATS	Automatic Train Supervision
BB	Building Blocks achievement
ERA	European Union Agency for Railways
EULYNX	European Initiative Linking Interlocking Subsystems
FM	Formal Methods
fUML	Foundational Subset for Executable UML Models
IC	Innovation Capabilities
IM	Infrastructure Managers
IP	Innovation Programme
L2TS	Doubly Labelled Transition System
LTS	Labelled Transition System
MAAP	Multi-Annual Action Plan
MBSE	Model Based System Engineering
OMG	Object Management Group
TD	Technology Demonstrator
UIC	International Union of Railways
UML	Unified Modeling Language
UNISIG	Union industry of signalling
WP	Work Package

3 Background

The present document constitutes the Deliverable D2.1 "Specification of formal development demonstrator" of Task 2.1 "Formal Development demonstrator prototype" of WP2 "Demonstrator Development for the use of Formal Methods in Railway Environment" of the project 4SECURail (GA 881775) in the context of the open call S2R-OC-IP2-01-2019, part of the “Annual Work Plan and Budget 2019”, of the programme H2020-S2RJU-2019.

The challenge to which 4SecuRail is deemed to deal, and its relation with the Shift2Rail Technology Demonstrator D2.7 "Formal methods and standardisation for smart signalling systems" is well described in the call S2R-OC-IP2-01-2019, as shown below:

Shift2Rail has identified the use of **formal methods** and standard interfaces as two key concepts to enable reducing the time it takes to develop and deliver railway signalling systems, and to reduce high costs for procurement, development and maintenance. Formal methods are needed to ensure correct behaviour, interoperability and safety, and standard interfaces are needed to increase market competition and standardization, reducing long-term life cycle costs.

To widen industry take-up of these key aspects, Shift2Rail plans demonstrating technical and commercial benefits of formal methods and standard interfaces, applied on select applications.

The industry survey performed in TD2.7 has identified the learning curve and uncertain cost/benefit ratio as obstacles: the decision to start using formal methods is deemed too risky by management. Shift2Rail proposes to define and prototype a demonstrator of state-of-the-art formal methods, including the use of standard interfaces, to address obstacles of learning curve and lack of clear cost/benefit analysis.

According to [MAAP2015, MAAP2017, MAAP2019] the Shift2Rail Innovation Programme 2 (IP2) will focus on innovative technologies, systems and applications in the fields of telecommunication, train separation, supervision, engineering, automation and security with a view to enhancing the overall performance of all railway market segments.
The Technology Demonstrator TD2.7 aims to contribute to the enabling of two Innovation Capabilities (IC) of the Shift2Rail Innovation Programme 2 (IP2):

- IC7 "Low Cost Railway"
- IC12 "Rapid and Reliable R&D Delivery"

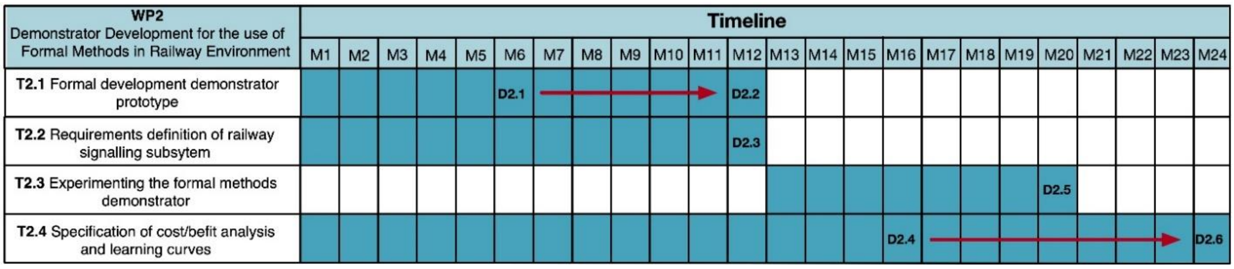
through the Building Block achievement BB2.7_1 "Formal and semi-formal methods for requirement capture, design, verification and validation, proposing open standards”.
4SECURail will contribute to the above Building Block achievement with the demonstration and evaluation of techniques based on formal methods to reduce life-cycle costs and improve the global reliability of the railway systems.

4 Objective/Aim

This Deliverable D2.1 reports the results of the first part of Task 2.1 "*Formal Development demonstrator prototype*" of Work Package WP2 "*Demonstrator Development for the use of Formal Methods in Railway Environment*".
The deliverable discusses the preliminary overall framework of the Demonstrator and identifies the selected choices for its specific architecture.
In the context of the Shift2Rail Multi-Annual Action Plan - Technology Demonstrator TD2.7 the development of the 4SECURail demonstrator falls within the Research Area "*Formal methods and standardisation*" and covers both activities:

- *"Demonstrate state-of-the-art formal methods for specification of requirements, automated design and software code creation"*
- *"Demonstrate improvements to high-level specification thanks to the use of semi-formal languages"*

FIGURE 1 from the project Grant Agreement, illustrates the workplan, with the expected deliverables and task interactions, for the Work Package WP2 "*Demonstrator Development for the use of Formal Methods in Railway Environment*", in which the activity reported in this deliverable is embedded.



WP2 Demonstrator Development for the use of Formal Methods in Railway Environment	Timeline																							
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22	M23	M24
T2.1 Formal development demonstrator prototype	▨	▨	▨	▨	▨	D2.1	→	→	→	→	→	D2.2												
T2.2 Requirements definition of railway signalling subsytem	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	D2.3												
T2.3 Experimenting the formal methods demonstrator													▨	▨	▨	▨	▨	▨	▨	D2.5				
T2.4 Specification of cost/befit analysis and learning curves	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	D2.4	→	→	→	→	→	→	→	D2.6

FIGURE 1 4SECURAIL TIMELINE FOR WP2

The objective of this deliverable is to define the rationale and the choices performed in terms of structure, methods and tools selection, for the definition of a semi-formal/formal software development process (Demonstrator) targeted to the construction of clear/rigorous/verifiable system specifications.
This defined demonstrator process will be exercised in the second part of Task 2.1 for the specification and analysis of the identified case study fragment. After any possible revision consequent to the experience gained during this first exercising of the demonstrator, the consolidated version of the demonstrator will then be used, as part of the activity of Task 2.3 ("*Experimenting the formal development demonstrator*"), for the analysis and verification of the full case study described in deliverable D2.3.
This final exercising of the consolidated demonstrator will be the basis for the study of the cost-benefit analysis of the approach and the evaluation of the learning curve for the use of the selected methodologies and tools that is part of Task 2.4 ("*Specification of cost-benefit analysis and learning curves*"), and that will be reported in deliverables D2.4 and D2.6.

5 Specification of formal development demonstrator

The purpose of this first deliverable of Task 2.1 is the description of the overall process that will be followed for the rigorous construction of system specifications (the formal methods demonstrator process), together with the suitability criteria for the supporting tools and the description of the architecture of the demonstrator itself.

In **Section 5.1** we clarify four points that play an important role in the correct framing of this effort. These points come from the constraints defined by the project objectives and by the relation of the 4SecuRail activity with respect to other complementary Shift2Rail projects. These points are related to:

- the role of formal methods
- the point of view of Infrastructure Managers
- the role of Standard Interfaces
- the issue of X2Rail2 [X2RAIL2] complementarity.

In **Section 5.2** we present the overall structure of the Demonstrator and the criteria for suitability of supporting tools. Two issues here are considered to deserve a more specific presentation:

- the role of UML/SysML ([OMG-UML], [OMG-SysML]) as standardised notation within the demonstrator.
- the role of the internally generated formal/semi-formal models with respect to the final system requirements specification that the demonstrator process is expected to define-

The overall generic structure described in this Section is independent from the specific case study of signalling system which it will be applied to.

In **Section 5.3** we describe the planned architecture of the Demonstrator, like the expected types of semi-formal and formal models that will be developed during the process, the possible types of properties that we might be interested to verify and the specific techniques for achieving that. This planned architecture does not include an experimental validation, as this will result from the remaining part of the activity of Task 2.1.

In **Section 5.4** the kind of input data that will be collected from the experimentation of the demonstrator process is outlined, as contribution to the activity of Task 2.4 (Costs/Benefits Analysis).

5.1 The reference framework

In this section, we discuss the main background needed to understand the rest of the deliverable. In particular, we present fundamental concepts related to formal methods, standard interfaces, the viewpoint of infrastructure managers and the complementarity issues with the X2Rail2 [X2RAIL2] project. Below, we first outline the role of each topic in the context of the 4SECURail project, and then we discuss each topic in separate sections.

The purpose of this deliverable is the description of a system development process. The focus of this effort is the exploitation and evaluation of the use of formal and semi-formal methods, with the goal of improving the quality of the generated artifacts and the reduction of their costs.
For this reason, in Section 5.1.1 we briefly overview what formal methods are and how they might impact the structure of our system development process.

The specification of the formal development demonstrator is based on the use case developed in Shift2Rail (X2Rail2) Deliverable D5.1, Section 5.4.1 "*Development of Systems with standardized interfaces*" [X2R2-D51]. In particular, with respect to that use case, we focus our effort on the same subject which is the exploitation of formal and semi-formal methods for the rigorous definition of system specifications that can be safely passed to multiple alternative developers. This subject is here summarised as “The point of view of the Infrastructure Managers”.
Section 5.1.2 illustrates in detail this aspect.

The same use case for the adoption of formal/semi-formal methods mentioned in [X2R2-D51] highlights the role of "standardized interfaces". In particular:

i) Shared, agreed, unique, standard interfaces for any signalling or control system, allows multiple producers to develop multiple, interoperable, fragments of the overall infrastructure in a robust and reliable way.
ii) Standard interfaces, described by means of a standardised notation, uniformly reduce the costs of creating and the difficulties of understanding interfaces specifications.

The role of standardized interfaces is better discussed in Section 5.1.3.

Finally, it is an important project objective to preserve and exploit the complementarity with respect to other Shift2Rail projects, and in particular X2Rail2. This issue is described in Section 5.1.4.

5.1.1 The role of formal methods

Formal methods refer to mathematically based techniques for the specification, development and verification of software and hardware systems [CENELEC EN50128].

In the following, when we use the general term formal method, we will implicitly include also semi-formal methods, i.e., those methods that use languages for which the semantics is not formally defined but depends on its execution engine. Furthermore, given that, in practice, a formal method always needs a support tool to be practically applicable, we will use the terms formal methods and formal tools interchangeably.

Formal methods have been largely experimented in industry for the development of safety-critical and mission critical products [WOD12]. Notable industrial cases on the usage of formal methods are the Maeslant Kering storm surge barrier control system [TWC01], where both the Z and the Promela formal notations have been used, and the Paris Metro on-board equipment [BBFM99], where the B-method has been employed. Transportation in general and railways in particular are domains in which formal methods have been largely experimented and applied [WOD12]. Research and industrial experiences concerning formal methods applications to railway systems' development have been published for more than thirty years [FAN13] [BGK18], and scientific publications in this field are increasing, showing that the interest in formal methods is still raising, but also indicating that more research is needed for a full industrial uptake of formal methods in railways.

Despite the quite long story of successful application of formal methods in the railway domain, it cannot yet be said that a single mature technology has emerged. Indeed, any proposed method or technique that goes under the umbrella of formal methods varies in its suitability and applicability to different stages of the signalling system development, and to different subdomains of railway signalling (interlocking, ATP, ATS, etc.). To this purpose, the ASTRail project [ASTRAIL] aimed at identifying, on the basis of an analysis of the state of the art, of the past experiences of the involved partners and on work done in previous projects, the candidate set of formal and semi-formal techniques that appear as the most adequate to be used in the different phases of the conception, design and development of railway signalling equipment.

Formal methods aim to guarantee, following some rigorous approach, the desired behaviour of a given computing system (see [AFPM11]). The notion of specification is central: a specification is a model of a system that defines its desired behaviour — what it actually should do, as opposed to how. A specification can vary for its level of abstraction, form the high level of abstraction of the desired properties of the system, to the more concrete level of an operational description of the behaviour of the system. In [AFPM11] these two problems are identified:

- the "model validation" problem: How to enforce, at the specification level, the desired behaviour?
- the "formal relation between specifications and implementations" problem: How to obtain, from a specification, an implementation with the same behaviour? Or alternatively, given an implementation, how can it be guaranteed that it has the same behaviour as the specification?

Different formal methods address these two problems in many different guises. Specifications may be analysed by animation/simulation, by transformation, or by proving properties. Implementations may be formally and mechanically derived from specifications in a correct-by-construction manner, or the former may be guaranteed to be correct with respect to the latter by different formal verification techniques, and under different formal correctness notions.

Basic Concepts

We introduce below some concepts and notions that characterise the application of formal methods and tools in the design of software systems that will be used in the following [AFPM11]

[OR17].

Model-based development puts a conceptual system model at the centre of the development process, from requirements engineering, through (model-based) design, to model-based testing -- which is characterised by test cases derived from models rather than from source code -- and possibly code generation -- which automatically translates system models to source code. Consistency among the models used in the various phases is ensured through model transformation and refinement. Model transformation can be seen as the automatic generation of a target model from a source model based on a transformation definition.

Refinement concerns the verifiable step-wise transformation of an abstract (high-level) formal specification into a concrete (low-level) executable program, such that each step increases the level of detail (e.g., which algorithm or which data type to implement).

Synthesis aims to automatically construct a system or program that is guaranteed to satisfy a given (high-level) specification.

Type checking offers a means to analyse the well-formedness of a model (or source code) with respect to its meta-model, which is formally specified as a type system that all models must conform to. This is a form of static analysis, i.e. check to be performed without executing the program/model.

Model checking is a technique to automatically and exhaustively verify whether a formal model of a system satisfies its specification, expressed as properties in a (temporal) logic. With respect to testing, model checking thus exhaustively verifies all possible behaviours, typically providing a counterexample in case a property is not satisfied. Affected by the *state space explosion* problem, that often jeopardizes its actual verification capabilities, model checking comes with a large variety of tools and techniques, as well as of notations to represent a system model, developed basing on different choices of basic principles, techniques and criteria:

- logical notations / algebraic processes / state-machine notations
- state based models (Kripke Structures) / event-based models (LTS) / mixed (L2TS)
- timed vs. untimed models
- probabilistic / statistical / nondeterministic models
- with limited data types (e.g. 1 .. 255) vs. with wide data types (e.g. int, real)
- explicit /symbolic /on-the-fly /bounded

Theorem proving is a deductive approach to prove the correctness of logical formulas by applying inference rules to them, either interactively or automatically, resulting in a proof script listing the deductive reasoning (for inspection by humans).

Model checking and theorem proving generally do not scale to huge systems. In such cases, (interactive) simulation (i.e., a sample path or execution) of the system model's behaviour can still provide valuable insights. Simulation tools can provide prototypes of software applications or tools, i.e., not yet complete versions of the software program under development, as is common for other engineering disciplines.

Abstraction and Nondeterminism

Formal models may need to *abstract* from details that are related to specific implementation choices or to very specific aspects of the system. Abstract specifications may include nondeterministic behaviour for modelling possible external interactions or internal choices, or may abstract away from aspects like time and data introducing further nondeterministic behaviours.

Depending on the process workflow, an abstract specification can be subsequently refined into a more detailed one, or a more detailed specification can be abstracted into a less detailed one to enable verification activities otherwise not possible.
In both cases we might have to deal with the problem of guaranteeing that the properties verified on the more abstract model are still preserved and satisfied by the more detailed one.

Executable/Simulatable/Verifiable models

For *executable* model we intend a system description that has the possibility of being executed, typically by automated code generation.

For *simulatable* model we mean an abstract system description whose behaviour can be simulated by a dedicated interpretation tool: a simulation run is typically played on simulation data.

For *verifiable* model we intend a system description at any level of abstraction on which formal verification of properties can be run (in reasonable time) by means of dedicated tools.

Typically, these three concepts can be related to different levels of abstraction of the system description. Simulatable models typically are defined at a higher level of abstraction than executable ones:

- an *executable model* behaves deterministically, as programs do; data queries and data transformations are defined by deterministic executable function;
- in a *simulatable* model system some details may not be completely defined and may lead to nondeterministic choices in the system behaviour. This nondeterministic behaviour can however be tested by automatic random animations or by interactively controlled animations. Clearly, executable (deterministic) models are also simulatable models;
- *verifiable* models may exhibit both deterministic and nondeterministic behaviour. In both cases, verification allows the automatic analysis of the whole system behaviour, and not just an interactive simulation of it.

All these three classes of models might all find their due place as part of the formal methods demonstrator because each of them allows to get a specific and useful view of the system being defined.

Test Cases Generation

Formal verification will never be able to fully validate the completeness and correctness of the specification with respect to intended user requirements. At most, a formal verification will be able to prove that the specific checked properties are indeed satisfied by the specification design.

Therefore, it makes sense, as part of the demonstrator process, to also include a testing activity that, starting from an executable/simulatable model of the system, allows to check the adherence of this model to the requirements, since testing is run at a lower abstraction level w.r.t. formal verification.

The tests, run in this activity, are derived from the model by means of test case generation facilities, and can be replicated on the final implementation of the product, to validate it against the requirements. The generated artifacts (e.g. testing suites) may be of interest also as an aid for a clearer understanding of the specification.

5.1.2 The point of view of Infrastructure Managers

As required by the project workplan, the project work stream 1 will take as reference the use case for the application of (semi) formal methods in the development of railways signalling systems defined in Subsection 5.4.1 "*Development of Systems with Standardised Interfaces*", of the deliverable D5.1 [X51] of the Shift2Rail X2Rail2 project.
The use case 5.4.1 deals with the adoption of formal methods from the point of view of the Infrastructure Managers.

The point of view of an Infrastructure Manager (IM) focuses on the “model validation” problem (see Sect. 5.1.1), since it has to provide a validated specification of a desired equipment to the Manufacturers.

In a classical client/developer scenario the common practice is the generation of - usually informal - system requirements document. This document can then be used by the developer to build an initial executable specification of the system, and then refine it (possibly using formal or correct-by-construction methods) into a final product.

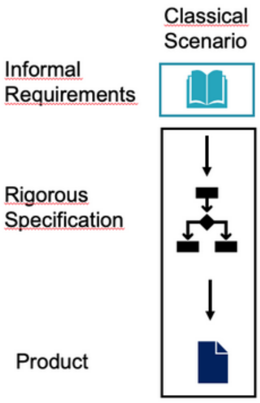


Figure 2 The classical Client-Developer scenario

The scenario in case of railway Infrastructure Managers is slightly different, since the main interest is on providing the same rigorous/verifiable specification not just to single developers, but to possibly multiple different developers that should produce equivalent products. This is precisely the case well described by the X2Rail2 use case selected as our reference, where defining a standard/rigorous/verifiable specification of the system to be developed becomes the IM responsibility.

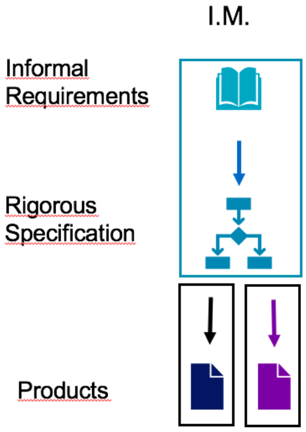


Figure 3 The Client-Multiple Developers scenario

Actually, in the case of railway Infrastructure Managers, the scenario is even more complex. In fact, the railway infrastructure is constituted by a multitude of subsystems (each one possibly developed by a different supplier) that must correctly interact among themselves.

In this case the problem of building rigorous/formal/verifiable specifications should extend also to the verification of the interactions between these components.

Clearly this does not hold only for *railway* Infrastructure Managers, but it is true also for any other kind of complex infrastructures (like, e.g. telecommunications).

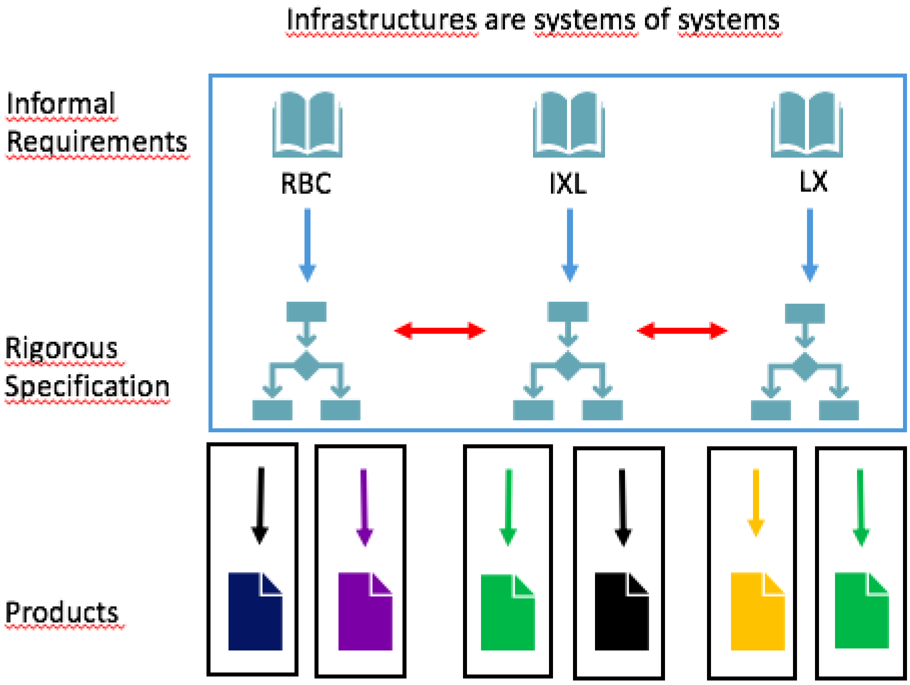


Figure 4 The Client – Multiple Developers scenario within a complex infrastructure

This introduces a further dimension of complexity. For example, safety properties can often be verified by reasoning at the level of single subsystems (e.g. ensuring that independently from the possible external interactions no unsafe conditions are even reached), but the same cannot be said for specific properties related to the composite behaviour of several subsystems (e.g. liveness, absence of deadlocks, or missing desired execution paths involving the behaviour of several subsystems).

A special case of these scenarios is when the produced specification takes the role of "standard specification" supported by international organizations (like UIC[UIC]/ERA[ERA]/UNISIG[UNISIG]), defined with the aim of creating interoperable railways in the whole Europe (Single European Railway Area, SERA).

5.1.3 The role of Standard(ized) interfaces

Our reference use case, described in Section 5.4.1 (*Development of Systems with Standardised interfaces*) of X2Rail2 D5.1, explicitly cites the EULYNX [EULYNX] *methodology* as a reference.

> **5.4.1 Development of Systems with Standardised interfaces**
> This use case is based on the EULYNX [11] methodology. Historically, infrastructure managers were supplied with monolithic systems, based on proprietary interfaces. A few years ago, a re-orientation of the means of production of future systems was initiated entailing purchasing modular systems. For example, an interlocking system comprises an electronic interlocking, a command and control system and field elements such as points, signals, and so forth. The fundamental concept of this new approach is to have these parts supplied separately. This

> requires standardised interfaces between subsystems and to adjacent systems, to enable different suppliers to supply compatible modules. This requires high quality specifications, as suppliers will be working with these blueprints and the infrastructure managers will carry out the system integration tasks.

Hence, "standardised interface” is intended to be a standard reference to be communicated to the suppliers by the Infrastructure Managers, with no dependencies on the way in which the interface is specified or the methodology through which the interface requirements specification has been generated.

Clearly, in the context of a project like EULYNX, whose purpose is that of rigorously defining all the interfaces of the Interlocking subsystem with all the other subsystems, it is perfectly reasonable to adopt a common methodology, language, and set of tools for achieving this purpose. This introduces the other meaning of "standard interfaces" as system interfaces described with a *standard notation*.

The goal of our demonstrator, from the point of view of the exploitation of "standard interfaces" is therefore twofold: a process exploiting the use of formal methods for the definition of standardised interfaces (goal: interoperability) described in standard notation (e.g. SysML) (goals: uniformity, understandability, non-ambiguity).

The EULYNX MBSE methodology is described in the document "EULYNX-Modelling Standard Eu.Doc.30" [EULYNXdoc30, Section 4], and summarised in Figure 5.

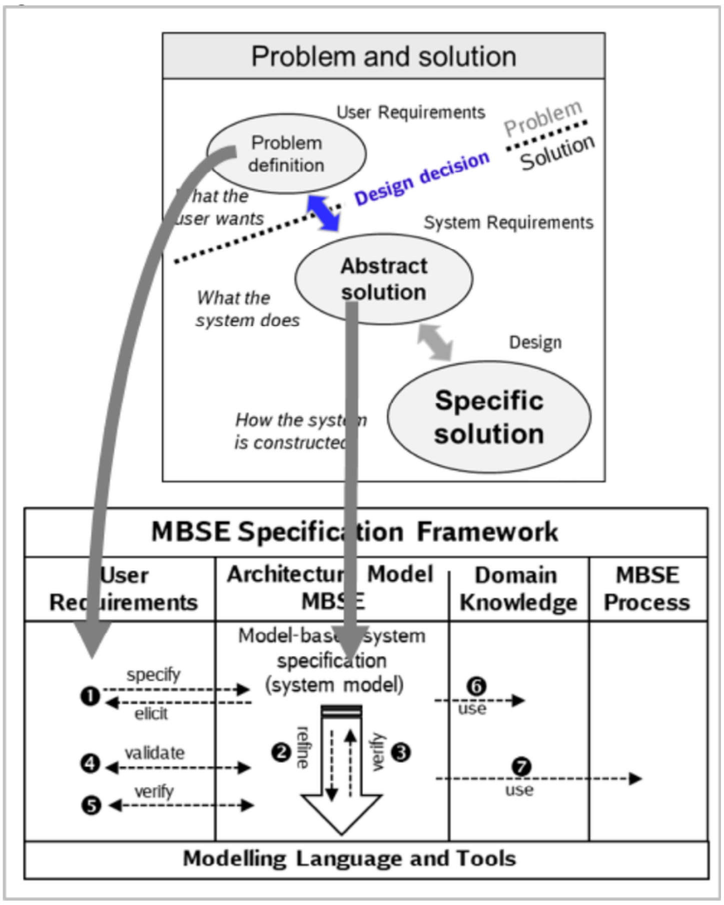


Figure 5 The EULYNX MBSE approach (Fig 1484 of Eu.Doc.30)

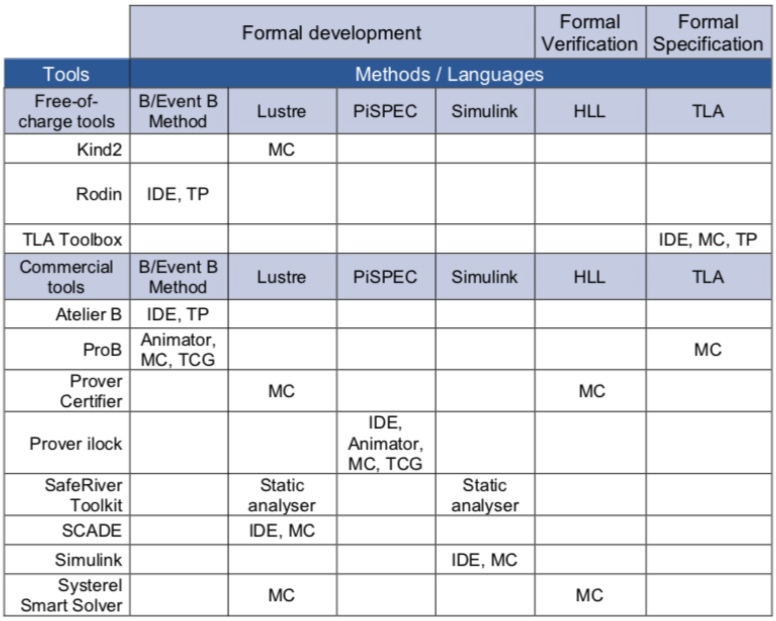
The overall approach being adopted for the Formal Methods Demonstrator, described in the subsequent Section 5.2, can be seen as a generalisation of the one adopted in EULYNX. The actual demonstrator specification can still make use of different specific tools, being this choice dependent on the kind of formal verification techniques that are being considered in the project, and on the specific project goal of exploiting formal methods for evaluating the costs/benefits of the chosen approach.

A common aspect which we believe it is important to try to preserve is the baseline adoption of UML/SysML as semi-formal model driven design methodology. This issue is discussed in more detail in Section 5.2.1.

A very detailed presentation of the expected benefits from the adoption of standard notations for standardised signalling interfaces can be found in [EIND].

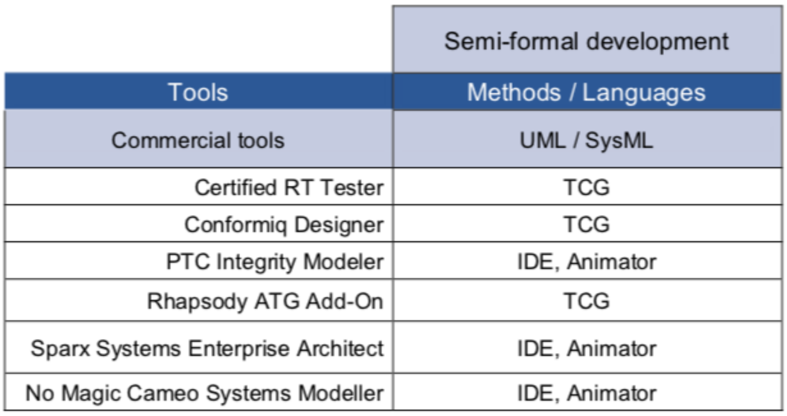
5.1.4 X2Rail2 complementarity

Task 5.2.1 of the X2Rail2 project also conducted a reasoned survey of the set of formal and semiformal methods proposed for use in a railway context. Although we are not constrained to use any of these tools/methodologies, surely this is a point which must be taken into consideration for the detailed design of our demonstrator. Figure 6 and Figure 7 show the formal and semi-formal methods and tools proposed by X2Rail2.



	Formal development				Formal Verification	Formal Specification
Tools	Methods / Languages					
Free-of-charge tools	B/Event B Method	Lustre	PiSPEC	Simulink	HLL	TLA
Kind2		MC				
Rodin	IDE, TP					
TLA Toolbox						IDE, MC, TP
Commercial tools	B/Event B Method	Lustre	PiSPEC	Simulink	HLL	TLA
Atelier B	IDE, TP					
ProB	Animator, MC, TCG					MC
Prover Certifier		MC			MC	
Prover ilock			IDE, Animator, MC, TCG			
SafeRiver Toolkit		Static analyser		Static analyser		
SCADE		IDE, MC				
Simulink				IDE, MC		
Systerel Smart Solver		MC			MC	

Figure 6 X2Rail2 Proposed formal methods tools (Table 1 of X2Rail D5.1)



	Semi-formal development
Tools	Methods / Languages
Commercial tools	UML / SysML
Certified RT Tester	TCG
Conformiq Designer	TCG
PTC Integrity Modeler	IDE, Animator
Rhapsody ATG Add-On	TCG
Sparx Systems Enterprise Architect	IDE, Animator
No Magic Cameo Systems Modeller	IDE, Animator

Figure 7 X2Rail2 Proposed semi-formal methods tools (Table 2 of X2Rail D5.1)

We have already mentioned the relevant role of the use case defined in Subsection 5.4.1 "Development of Systems with Standardised Interfaces" of the deliverable D5.1 of the Shift2Rail X2Rail2 project, which refers to the adoption of formal methods from the point of view of the Infrastructure Managers.

Figure 8, extracted from the mentioned D5.1 of X2Rail2, shows a possible workflow of the system development process, based on formal and semi-formal methods, as it might be used by Infrastructure managers to build rigorous and verifiable specifications (system requirements) to be delivered to different developers for their development.

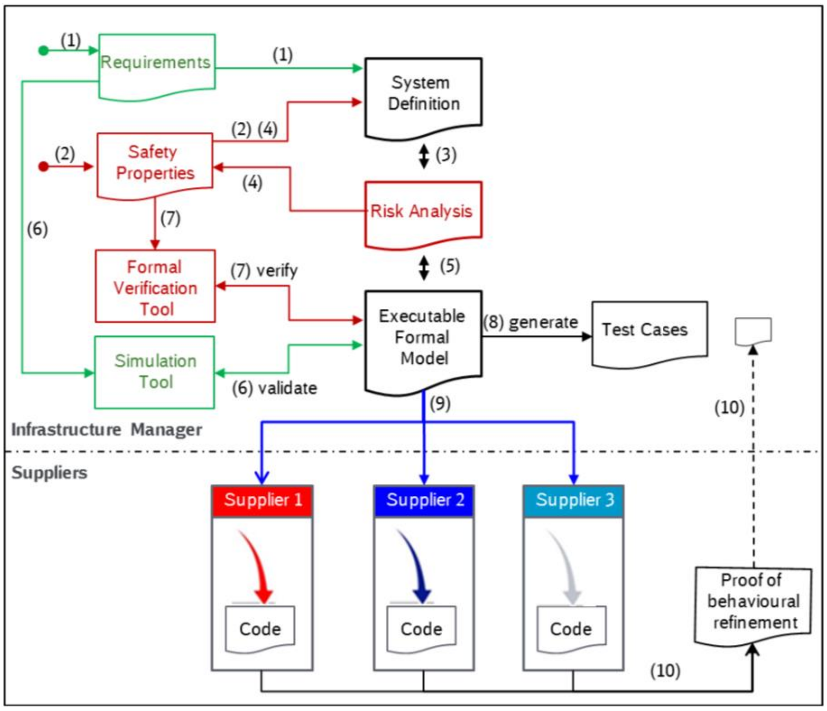


Figure 8 X2Rail2 Use case for formal development of systems with standardised interfaces

Where, in particular, it is described that:

> The specification process starts with the system definition based on requirements derived from stakeholder needs (1) and regulation-based safety properties (2):
> The system definition comprises basically the technical system context and the functional system context, defining the interfaces of the system and the information flows at them.
> The system's use cases – the services the system is expected to perform for its environment – are described by scenarios which order the defined information flows in time, and thus specify the expected externally observable input/output behaviour of the system at the upper level of abstraction.
> The system definition is described using the Systems Modeling Language (SysML) [10], a semi- formal graphical modelling language. A detailed description of the methodology to model the system definition is given in the EULYNX Modelling Standard [12].
> With the system definition as basis the risk analysis is carried out (3). An analysis of the different types of possible hazards is made and hazard-based safety properties derived. They supplement the pre-existing regulation-based safety properties (4) and are used to adjust the system definition if necessary.
> Based on the externally observable input/output behaviour defined in the system definition phase, including relevant results of the risk analysis, an executable model of the externally observable behaviour is created (5).
> The executable model is used for validation (6) of the requirements by simulation (virtual prototype), formal verification of the safety properties (7) and automated generation of test cases (8).
> The semi-formal model is given as part of the tenders to the suppliers (9) which respond with the proof that the behaviour of their implemented system is a refinement of the specified one (10).

The above generic workflow is clearly well applicable, with the needed variations, also in our case, and in Section 5.3 “The architecture of the 4SECURail demonstrator" the main differences between this workflow and our Demonstrator structure are illustrated.

5.2 The overall structure of the demonstrator process

In this section we describe the overall structure of a demonstrator process aimed at employing formal methods to support infrastructure manager. The next section will be dedicated to the specific instantiation of this process in the 4SECURail context.

The overall structure of a generic Software development process targeted to the definition of rigorous system specifications which exploits the use of formal methods (our Demonstrator) can be described as in the Figure 9.

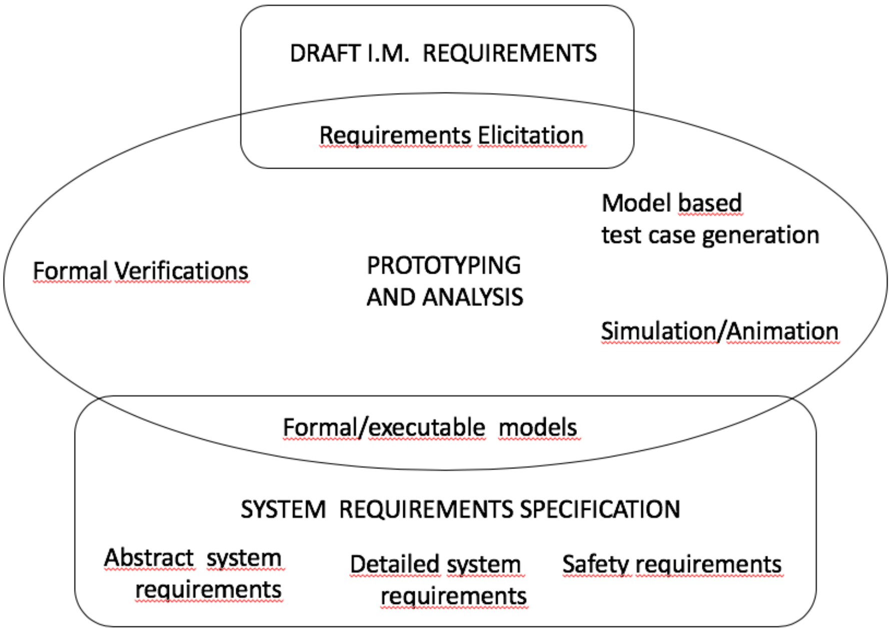


Figure 9 Overall generic structure of demonstrator (first case)

First Case (with requirements elicitation). Starting from some input describing the initial I.M. requirements of the system, we start an agile (in the style of [AGILE]) development phase in which the requirements are transformed into "formal/executable models". These models are developed incrementally, and continuously analysed by means of formal verifications, simulations, animations, and collecting test cases for documentations.

These abstract formal models can also be refined by adding additional details into "refined executable models" that may help in validating the system behaviour possibly through simulations and animations.

Once these formal models are sufficiently stable, they represent the base for the generation of the demonstrator output (the official system requirements specification), in the form of description of "abstract system requirements", "safety requirements", "detailed system requirements". The generated system requirements are still likely to be expressed in natural language but enriched with tables and diagrams extracted from the formal/semi-formal models. The formal/semi-formal models themselves might be made available as complementary documentation.

From one side, while the generation of multiple, different semi-formal / executable / simulatable / formally verifiable models allows to get a deep understanding of the system design from many points of view and many levels of abstractions, from the other side this multiplicity raises the problem of keeping these models somewhat "synchronized". E.g. if, for some reason, one of the models needs to be modified because of the discovery of some defect, the impact of the change on the other models surely cannot be ignored. This may require the generation and maintenance of some kind of cross-references between these artifacts, and probably also between these artifacts of the final "system requirements specification" resulting from the process. The effort needed for keeping all the different artifacts well synchronised should not be underestimated and might play a non-trivial role in deciding how many "points of view" to take into account.

Second Case (without requirements elicitation). The whole schema still holds in the case in which the input of the overall Demonstrator process is not constituted by *Draft I.M. Requirements*, but by an already consolidated/official set of system requirements / safety requirements, that should be the object of more rigorous analysis.

In this case we simply would not have the *Requirements Elicitation* activity oriented to the consolidation of the *Draft I.M. Requirements,* see Figure 10. The difference in the wording "PROTOTYPING" versus "MODELLING", in this second case, just reflects that if the starting point is an already consolidated specification, the modelling activities (in terms of tools and methods) might be somewhat different from the incremental prototyping activity driven by a rigorous/formal Requirements Elicitation phase.

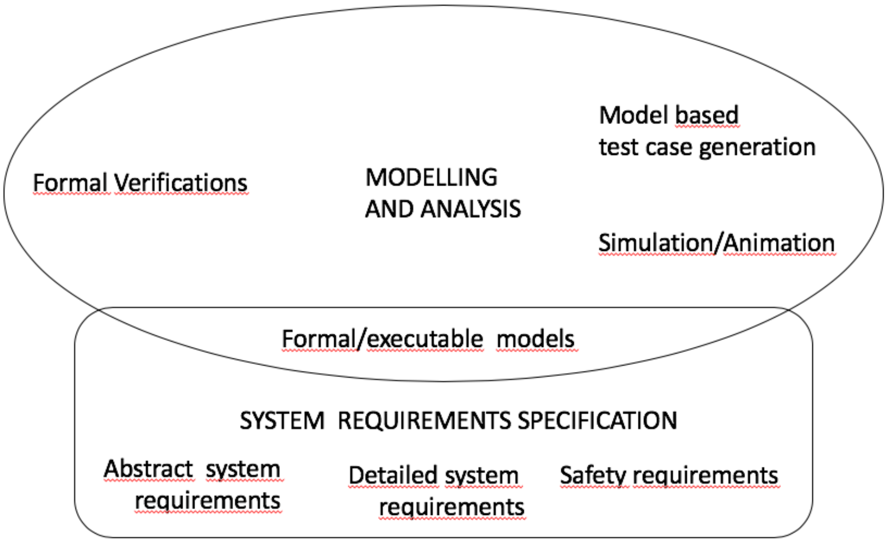

Formal Verifications
MODELLING AND ANALYSIS
Model based test case generation
Simulation/Animation
Formal/executable models
SYSTEM REQUIREMENTS SPECIFICATION
Abstract system requirements
Detailed system requirements
Safety requirements

Figure 10 Overall generic structure of demonstrator (second case)

Third Case. The same overall schema might also work in the mixed case in which an already consolidated set of system requirements/safety requirements might have to be extended/updated by an additional set of new user requirements (somewhat of composition of the previous two cases). In these cases, the availability of previous formal/executable artifacts would be of great help for the process.

We consider as already acknowledged (see for example the related Shit2Rail surveys in [X2R2-D51, ASTRAIL-D41, ASTRAIL-D43]), that there is not a single formal method or tool that can fit all the possibly desired verification and modelling needs in the railway field. Therefore, the whole *Modelling and Analysis* activity is supported at its best by a rich integrated ecosystem of tools and methodologies, rather than a single monolithic, usually closed, tied to single specific methodologies, framework. We recognize, however, that at least in the first case, where a classical V shaped process might be followed covering all the steps from *Requirement Elicitation* to *Official Requirements Specification generation and verification*, a reference modelling frameworks might actually help in building and maintaining all the documentation related to the various artifacts being generated.

5.2.1 The role of UML / SysML

UML (Unified Modeling Language) is a standardized modeling language consisting of an integrated set of graphical diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems [WHATISUML].

UML, in its SysML version, has been adopted also in the EULYNX project within its underlying methodology for the development of standard interfaces. A detailed analysis of this approach is well described in [EIND].

Graphical designs do often convey information to the reader with a wider band than just text and require less effort in the reader for receiving it.
However, a textual representation readable/writeable by humans is equally important for the simpler way in which it can be produced, shared, translated, modified, and communicated.
We believe that both kinds of representation should be made available, and they should be and remain in synch.

It is also important for the designer to be able to simulate the UML behavioral models (e.g. state machines) to have some initial feedback on the correctness of the design with respect to the intended requirements. Otherwise models risk being precise, but wrong.

A prerequisite for a reasonable introduction of UML as reference notation inside a formal methods Demonstrator process is that the meaning of the UML designs shall not be ambiguous or uncertain. Since its origins, this has been recognised as a major problem for some of the behavioural diagrams of UML like state machines.

The main recognised problems with this behavioural notation are in fact (see e.g. [FSKR29, SG30]:

- Uncertainties in the semantics
- Absence of standard action language
- Lots of implementations freedoms

Several studies and proposals have been conducted in the recent years with the goal of associating a formal semantics to the UML behavioral diagrams (see e.g. [CD2007, BCDRS, L2013]), but none of these actually succeeded in solving the problems.
An important step forward to overcome this problem has been done by OMG (Object Management Group) with the standardization of fUML (*Foundational Subset for Executable UML Models*). [OMG-fUML1], which is also associated with an official reference implementation [OMG-fUML2].
This definition of fUML is complemented with the definition of textual syntax for its action language ("*Alf*" [OMG-Alf]), and by the definition of the "*Precise Semantics of UML Composite Structure (PSCS)*" [OMG-PSCS].

The purpose of this fUML effort is precisely the one of defining an initial subset of UML which is free from the semantic uncertainties affecting the full standard and that might define a rigorous Model of Computation for the UML behavioral diagrams.

The remaining limits of this effort is that this fUML definition is still described in natural language, and that the "reference implementation" (that might play the role of non-ambiguous operational semantics) is currently being implemented only with respect to activity models [OMG-fUML2]. The Alf definition itself, when considered in conjunction with the state machine notation, is currently defined just through an "Informative Annex" [OMG-Alf] with no normative role.
More details on fUML are provided in Annex 8.2.

W.r.t. our overall demonstrator process UML can play three different roles:

- as complementary graphical documentation of specific aspects of the system requirements definition.
- as a direct notation for the execution and simulation of system models.
- as baseline for translations towards other formal notations supported by strong verification capabilities.

The use of UML for system design and documentation is supported by an extremely rich set of tools, partially reported in Annex 8.1. If we are interested in just designing diagrams for complementing the natural language description of a system, we might find useful to use UML tools exploiting more immediate and user-friendly textual encoding of the diagrams (like Umple, textUML, et al., cited in Annex 8.1).

Support for the use of UML for execution/simulation of the system behaviour is much more limited and constrained to a handful of alternatives, equally reported in Annex 8.1.

None of the "industry ready" UML tools allows a direct verification of behavioural models; as far as we know, only a few academic prototypes (e.g. UMC [UMC1, UMC2]) have been developed precisely to this purpose.
Therefore, we are only left with the possibility of performing the translation from the UML models into other formal notations supported by verification frameworks.
In the literature there are plenty of papers describing experiences in this kind of translation [see e.g. [PEML, GMK2012, CFLW, BBJTD2018, F2008, YLWD, NPS2009, BR2010, Y2010, SSB2012, RBS2019, HKLMPMS, KMR2002, CC2004, BFMMMNV, OD2017, OSG2004, JDJLP] but none of them seem to have been well supported and integrated inside "industry ready" UML frameworks.

Given the focus of the 4SECURail demonstrator on formal methods, the last described use of UML (baseline for translations) is probably the one that is more tied to the project goals, even if also the other two uses (documentation and simulation) may play a relevant role inside the Demonstrator.

From this point of view our preferred choice would be the use of an even stricter subset of the fUML state machine diagrams, defining a very simple state machine structure that would allow a direct translation into the main formalisms adopted by verification and simulation tools, such as Event-B [EVB] / LNT [GLW2017] / Uppaal [UPPAAL].
Notice that we do not have the goal of defining a subset valid in the general case, but we just explore this approach in our limited case-study, because we believe that this point of view is worth a demonstration and experimentation.

We can observe that EULYNX gives a precise information of the specific tools and methodologies adopted in the project, like

- Atera as action language for behavioral diagrams
- PTC [PTC-Windchill] as a graphical design and animation tool for the specifications.

In our case the criteria for selecting specific UML/SysML tools might possibly lead to a different choice, that will be based on the following considerations:

- The non-ambiguity and standard-quality of the supported notations,
- The openness of the framework - i.e. how easy it is to import/export/translate the notations versus other frameworks,
- The usability of the tool user interface,
- The degree of support for nondeterministic aspects in the design and
- The degree and cost of support and training for the clients.

5.2.2 The expected output of the demonstrator process

The set of artifacts in output from the formal methods demonstrator process are represented in our overall generic model by the final "System Requirements Specification". Actually, these artifacts might be of different nature and with different purposes:

- A rigorous natural language textual description, possibly enriched with standard diagrams and tables, that may constitute the legal document associated to the specification;
- A simulatable semi-formal system description: this artifact might be considered as a very useful complement that might be made available to the developers for checking their correct understanding of the system to be developed;
- Formal verifiable specifications, allowing the developers to possibly exploit these models for "correct by construction" code generation, and allowing the Infrastructure Managers to maintain, further verify, and possibly improve the System Specification itself;
- A set of tests generated and successfully applied for the analysis of the various models, that can provide developers with guidance and early verification for the testing of the ongoing product development.

5.3 The architecture of the 4SECURail demonstrator

There are four points that directly affect the definition of the architecture of the demonstrator:

- In which way the semi-formal models describing the system requirement specification are generated for being analysed?
- In which way the simulatable/executable models of the system are generated?
- In which way the formal models of the system are generated and verified?
- In which way the case study selected for the exercising the demonstrator may affect its architecture?

The following paragraphs give more details on all these aspects.

Specification with standard notations

We believe it is important to adopt as reference inside the demonstrator a standardised description of systems specification which, considering also the indications coming from the EULYNX and X2Rail projects, are based on UML/SysML diagrams, and in particular on behavioural diagrams (state machines and sequence diagrams).

The ideal (imaginary) approach to system specification should rely on an advanced support framework allowing to generate clear, graphically appealing, rich of content, possibly interactive, diagrams. Starting from these, interactive simulation to explore the possible nondeterministic alternatives present in the behaviour would be possible, allowing the formal verification of system properties.

Unfortunately, this ideal approach is still very far from the current state of the art. In practice, if we really want to generate clear, graphically appealing, rich of content, diagrams, it is necessary to make use of specific drawing-oriented tools (e.g. in ASTRail, Graphviz [GRA] has been used for this purpose) that do not support simulation and verification.

Instead, diagrams automatically generated by UML/SysML-based frameworks are often of a not sufficient graphical quality and may not contain all the useful detailed information (e.g. the abstract events that relate a system transition to one or more system requirements). At the same time, however, they may be directly used to perform simulation and verification.

The use of UML/SysML-based frameworks allows the progress from the system design to code generation in a rather smooth way. This usually is of interest of developers but of less interest for the point of view of I.M.

It is therefore likely, unless more experience comes out from the actual demonstrator experimentation, that a graphical SysML design is adopted in our demonstrator without any predetermined relation with specific UML /SysML -based framework.

Frameworks for Executable / Simulatable Modelling

As already described in Section 5.2.1 the UML/SySML state machine descriptions might be exploited in the demonstrator not only as graphical designs with documentation purposes, or as basis for translations info formal verifiable notations, but also as simulatable models suitable for experimenting the actual system behaviour.

This kind of use requires the exploitation of much more complex (to learn, to use, to acquire) frameworks supporting execution and simulation of composite systems based on interacting state-machines. The survey on semi-formal tools conducted by X2Rail2 and presented in D5.1 (see X2Rail Table 2 reported in Section 5.1.4) indicates as possibly recommended frameworks for system simulation the following ones:

- PTC Integrity Modeler (now Windchill Modeler SySim) [PTC-Windchill]
- Sparx Systems Enterprise Architect [SPARX]
- No Magic Cameo Systems Modeller (now Dassault 3DS Cameo Systems Modeller) [3DS]

It is not sufficient to look at the available online documentation for the various frameworks to identify the best solution, in the context of our demonstrator, as possibly recommended frameworks for system simulation.

Therefore, we will defer this choice to the prosecution of Task 2.1, after a hands-on experimentation of the various possibilities with the selected initial fragment of the chosen case study.

In the context of the 4SECURail demonstrator the exploitation of a framework allowing to directly simulate the designed behavioral models in agreement with the official OMG fUML semantics would be a great contribution because it would allow to ensure that the designed graphical models actually reflect in a not ambiguous way the expected system behavior.

Formal Verification by Model Checking

Independently from the kind of tool support for the generation (and possibly simulation) of UML/SysML state-machine designs, our main goal is to transform these standard UML/SysML designs into verifiable formal models.

Theorem proving and Model Checking can probably be considered the two most used approaches to system verification, also in railway related contexts.
However, Theorem proving, e.g. as supported by Atelier B, seems more fitting a specification refinement process that guides the correct-by-construction generation of code starting from an initial formal design. Model checking instead seems more fitting a model-based approach in which a simulatable design is explored and verified in all its possible evolutions. In 4SECURail we follow the model checking approach since we are not interested in code generation.

In particular we will take advantage of the experience gained with the ASTRail project (see [ASTRAIL-D43]), where UML state machine descriptions were translated into EventB state machines and subsequently analysed and verified by model checking with the ProB tool [PROB].
ProB is an animator and model checker for the B-Method. It allows animation of many B specifications and can be used to systematically check a specification for a range of errors. ProB is one of the tools also recommended by X2Rail2 for formal verifications (see X2Rail2 Table 1 in Section 5.1.4). Some of the reasons for the successful experience of its use in ASTRail project and to reuse it also in 4SECURail are the following:

- It is a free, open source product whose code is distributed under the EPL v1.0 license [http://www.eclipse.org/org/documents/epl-v10.html]
- Is actively maintained and commercial support is available from Formal Mind [http://www.formalmind.com/]
- Runs on Linux, Windows, and MacOS environments
- It has several nice, very usable graphical interfaces, but can also be used from the command line
- It is well integrated in the B / EventB ecosystem (Rodine, Atelier B, iUML, B Toolkit)
- It allows construction, animation and visualisation of nondeterministic systems
- It allows formal verifications through different techniques like constraint solving, trace refinement checking, model checking.

There are also known weak points related to its use, which in our case are:

- Does not allow the explicit modelling of multiple mutually interacting state machines. The only way to achieve that is to merge all the separate machines into a global one.
- EventB state machines are different from UML/SysML state machines. At the current state of art several proposals of translations from UML to ProB state machines have been made, but no industry-ready product currently supports that mapping.
- Model checking does not support compositional approaches based on bisimulations which are congruences with respect to parallel composition operations. In simpler words the verification approach does not scale when the system is composed by many mutually interacting asynchronous state-machines.

More details of the ProB tool are reported in Annex 8.3.

Modelling the behavior of a system through the design of a single state machine has the advantage that this design can often be translated into the notations supported by formal verification frameworks with a reasonable effort.

However, if we have to verify properties that depend on the behavior of more interacting asynchronous systems, the situation becomes more difficult. If the components are not too complex, or not too many, a possibility is to merge all of them into a unique "global" system modelled again as a single state machine. If the various system components are too complex, or too many, this approach risks however incurring in the problem of state explosion.

In this case we can imagine two types of solutions:

- One solution is to constrain the verification to a rich set of scenarios. I.e. not verifying the system in its complete variability, but only under certain assumptions (like for example, absence of fatal errors in certain components, only one/two/three trains moving from one RBC to another, limited presence of communication errors, just to mention some).
- The other solution is to exploit alternative formal notations historically oriented towards the design and verification of asynchronous interacting systems and supported by specialised theoretical basis like process algebras (see e.g. [MCRL2, CADP, FDR4]).

We are unable at the current time to evaluate the overall final complexity of the chosen case study, and if model checking within the ProB framework will be sufficient to verify overall systems constituted by interacting components. In any case our approach does not prevent the experimentation with alternative translations towards verifications engines more oriented to the analysis of "parallel asynchronous systems".

The three aspects described above are summarised in Figure 11.

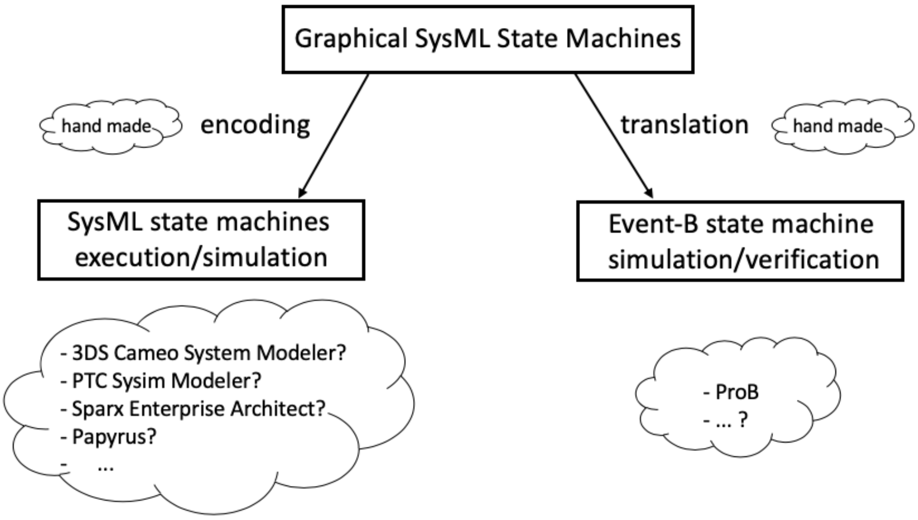

Graphical SysML State Machines
hand made
encoding
translation
hand made
SysML state machines execution/simulation
Event-B state machine simulation/verification
- 3DS Cameo System Modeler?
- PTC Sysim Modeler?
- Sparx Enterprise Architect?
- Papyrus?
- ...
- ProB
- ... ?

Figure 11 Execution flow of the demonstrator prototype

The case study

The case study to test the formal methods demonstrator proposed by 4SECURail is the RBC/RBC protocol, as specified by the UNISIG RBC/RBC Handover [SUB-039] and Safe Communication Interface [SUB-098].
A Handover procedure is needed to manage the interchange of train control supervision between two neighbouring RBCs. When a train is approaching the end of the area supervised by one handing over RBC, an exchange of information with the (new) accepting RBC takes place to manage the transaction of responsibilities. RBC/RBC interface is a typical product where development processes of different supplier meet, and is therefore an optimal choice to investigate how natural language specification may create the possibility of diverging interpretations, leading to interoperability issues. The details of the case study and the rationale for this choice will be described in Deliverable D2.3 of Task 2.2.
Being UNISIG SUBSET-039 and SUBSET-098 already consolidated standards, the overall structure of our demonstrator process will reflect the second point of view of those described at the beginning of Section 5.2 and illustrated in Figure 10, which is the case of formal methods demonstrator process used for just analysing, verifying, and possibly improving an already existing standard specification.

With respect to the X2Rail2 workflow shown in Section 5.1.4 - Figure 6 we can say that a Risk Analysis phase is not needed in our case study because safety threats have been already addressed into SUBSET-039 and SUBSET-098. Additional safety requirements will be added if required by the specific modelling of the system.

With respect to the same workflow of Figure 6 also the test generation subphase has a different flavour, because in our case it is not oriented towards the final validation of the developed products, but towards the achieving of a further degree of confidence on the correctness of the generated models, especially w.r.t. those aspects not covered by formal verifications. This test generation subphase might in fact evolve within the semi-formal SysML simulation framework (if the selected tools actually support it), that might describe the system at a different level of abstraction with respect to the verified formal models (e.g. modelling in more detail some data-related and time-related aspects).

The output, in terms of artifacts, of our demonstrator process will reflect the structure described in Section 5.2.2.
In our particular architecture, being the input requirements an already stable official UNISIG standard, we will not need to rewrite it using again a natural language notation, even in the case the rewriting could appear as more precise or complete. We can however complement it with annotations, if found useful, and/or enrich it with further artifacts developed with the demonstrator process, such as SysML models, animatable modes, formal model, test cases, and the needed cross references among these components.

5.4 Inputs for the cost-benefit analysis and learning curve evaluation

During the experimentation of our demonstrator process with its application to the selected case study, both in the second Part of Task 2.1 (initial fragment) and in Task 2.3 (full case study) it will be important to assess as much data as possible on the costs and cost categories embedded in the proposed approach.

The goal will not be to record time-related costs for the demonstrator development, but to identify cost categories which are likely steering the development of a generalised system having the features of the demonstrator. Costs categories may be preliminarily clustered as:

- costs for acquiring tool licences (either based on the actual costs incurred for licences necessary for the demonstrator or costs of the full commercial licence for the same tool, including commercial support and training), or cost of licence for alternative tools with respect to the ones used in the demonstrator.
- time-related costs for research and development: such costs are dependent on the estimation of effort (person-days) needed to learn a specific tool and methodology (entailing the learning curve of FM for the system suppliers), and to the time/effort needed to generate the animatable SysML specification, to generate the formally verifiable models, to select, design and perform the verifications of the properties of interest, to maintain the various model well synchronized.

Before being actually usable for the costs-benefit analysis, this effort data might have to be adjusted for taking into account the bias resulting from the previous already existing competences and knowledge of the involved people, and will need to undergo a benchmark with literature sources.

As a methodological pillar, the cost-benefit analysis will analyse costs and benefits associated to the exploitation of formal methods, against the baseline scenario, which does not foresee the adoption of formal methods. The outcome will be the differential of costs and benefits associated with the generalised adoption of the system described by the demonstrator. The assessment of baseline costs (either for licences or time-related costs) will be estimated and reported in D2.4.

6 Conclusions

The activity of the formal methods demonstrator process will start with the definition of the SysML designs describing the UNISIG RBC Handover system selected as case study.
After this point the process will fork, experimenting from one side the inclusion of the designs in a MBSE framework for subsequent animation and generation of use cases and test cases of interest, and from the other side experimenting the translation of the SysML design into formal Event-B state machines for subsequent formal verifications with ProB.
The resulting flow is depicted in Figure 11.

The specific MBSE tool that will be used for SysML simulation will be selected (if at least one found satisfying all our needs) during the initial experimentation of the demonstrator in Task 2.1 (second part) that will define the final demonstrator prototype structure (Deliverable D2.2 - November 2020).

The actual contribution of this deliverable goes far beyond the final answer to the question about "which tools and methods will be actually used by the demonstrator", but it consists also in the reasoning and the rationale that have led to the selected choices. In particular three important issues deserved a specific analysis and discussion:

- The clarification of the usefulness of formal methods from the point of view of the Infrastructure Managers,
- The relations between our demonstrator and other relevant projects like EULYNX and X2Rail2, and
- The role that the semi-formal SysML notation should play within our formal methods demonstrator process.

It is important to remark that the 4SECURail Demonstrator does not have the goal of identifying "the best set of formal methods and tools to be used in a railway context". This investigation has just the specific goal of conducting an experiment to demonstrate the use of formal methods for the construction of robust, reliable system requirements specifications, and observing, extrapolating, and analysing the experience gained from it.

It is also important to remark that the use of formal methods analysed in this project is not the kind of use that might be done by system developers for the production of correct, robust and verifiable systems, even if the developers might surely take advantage of the additional level of rigor in the generated requirements specifications and accompanying artifacts, for a more immediate understanding and (possibly formal) generation of the product.

7 References

[3DS] 3DS Catia nomagic "Cameo Systems Modeler"
https://www.nomagic.com/products/cameo-systems-modeler

[AFPM11] Bacelar Almeida J., Frade M. J., Sousa Pinto J. & Melo de Sousa S (2011). "An Overview of Formal Methods Tools and Techniques in Rigorous Software Development -An Introduction to Program Verification". Undergraduate Topics in Computer Science, Springer, 15--44.

[AGILE] The Agile Alliance https://www.agilealliance.org

[ASTRAIL] ASTRail project. https://projects.shift2rail.org/s2r_ip2_n.aspx?p=ASTRAIL

[ASTRAIL-D41] ASTRail Deliverable D4.1 "Report on Analysis and on Ranking of Formal Methods" http://astrail.eu/download.aspx?id=bb46b81b-a5bf-4036-9018-cc6e7d91e2c2

[ASTRAIL-D43] ASTRail Deliverable D4.3 "Validation Report" http://astrail.eu/download.aspx?id=d7ae1ebf-52b4-4bde-b25e-ae251fd906df

[BBFM99] Behm, P., Benoit, P., Faivre, A., & Meynadier, J. M. (1999). "METEOR: A successful application of B in a large project". In International Symposium on Formal Methods (pp. 369-387) Springer, Berlin, Heidelberg.

[BBJTD2018] Valentin Besnard, Matthias Brun, Frédéric Jouault, Ciprian Teodorov, Philippe Dhaussy "Unified LTL Verification and Embedded Execution of UML Models" MODELS '18: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems October 2018 Pages 112–122 https://doi.org/10.1145/3239372.3239395

[BCDRS] Broy M., Crane M.L., Dingel J., Hartman A., Rumpe B., Selic B. (2007) "2nd UML 2 Semantics Symposium: Formal Semantics for UML." In: Kühne T. (eds) Models in Software Engineering. MODELS 2006. Lecture Notes in Computer Science, vol 4364. Springer, Berlin, Heidelberg https://link.springer.com/chapter/10.1007/978-3-540-69489-2_39

[BFMMMNV] S. Bernardi, F. Flammini, S. Marrone, N. Mazzocca, J. Merseguer, R. Nardone, V. Vittorini "Enabling the usage of UML in the verification of railway systems: The DAM-rail approach" Reliability Engineering and System Safety 120 (2013) 112–126

[BGK18] ter Beek M.H., Gnesi S. and Knapp A. (2018). "Formal methods for transport systems". International Journal on Software Tools for Technology Transfer, Springer, (pp 237—241)

[BR2010] P.Bhaduri, S. Ramesh "Model Checking of Statechart Models Survey and Research Directions" https://arxiv.org/pdf/cs/0407038

[CADP] CADP website https://cadp.inria.fr

[CC2004] Chen J., Cui H. "Translation from Adapted UML to Promela for CORBA-Based Applications" In: Graf S., Mounier L. (eds) Model Checking Software. SPIN 2004. Lecture Notes in Computer Science, vol 2989. Springer, Berlin, Heidelberg

[CD2007] Michelle L. Crane · Juergen Dingel "UML vs. classical vs. Rhapsody statecharts: not all models are created equal" Softw Syst Model (2007) 6:415–435 doi: 10.1007/s10270-006-0042-8

[CENELEC EN50128] EN 50128:2011 "Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems". CENELEC CLC/TC 9X standard, 2011-06.

[CFLW] Caltais G., Leitner-Fischer F., Leue S., Weiser J. “SysML to NuSMV Model Transformation via Object-Orientation". In: Berger C., Mousavi M., Wisniewski R. (eds) Cyber Physical Systems. Design, Modeling, and Evaluation. CyPhy 2016. Lecture Notes in Computer Science, vol 10107.

[DBLM2002] V. Del Bianco, L. Lavazza and M. Mauri, "Model checking UML specifications of real time software," Eighth IEEE International Conference on Engineering of Complex Computer Systems, 2002. Proceedings., Greenbelt, MD, USA, 2002, pp. 203-212, doi: 10.1109/ICECCS.2002.1181513.

[EIND] Bui, N. L. (2017). “An analysis of the benefits of EULYNX-style requirements modeling for ProRail”. Eindhoven: Technische Universiteit Eindhoven. https://research.tue.nl/en/publications/an-analysis-of-the-benefits-of-eulynx-style-requirements-modeling

[ERA] European Union Agency for Railways https://www.era.europa.eu/

[EULYNX] The Eulynls project site. https://eulynx.eu/

[EULYNXdoc30] EULYNX-Modelling Standard Eu.Doc.30 v3.0 (0.A)

[EVB] Event-B.org Website http://www.event-b.org/

[F2008] Critical Software S.A. (slides) “Model-Checking and Validating UML Models: Current Capabilities and Limitations” ESA Workshop on Avionics Data, Control and Software Systems (ADCSS) https://distrinet.cs.kuleuven.be/projects/evolve/public/publications/02_01_Faria.pdf

[FAN13] Fantechi, A. “Twenty-five years of formal methods and railways: what next?” In International Conference on Software Engineering and Formal Methods (pp. 167-183). Springer, Cham.

[FDR4] FDR4 The CSP Refinement Checker website https://cocotec.io/fdr/index.html

[FSKR29] Fecher H., Schönborn J., Kyas M., de Roever WP. (2005) 29 “New Unclarities in the Semantics of UML 2.0 State Machines” In: Lau KK., Banach R. (eds) Formal Methods and Software Engineering. ICFEM 2005. Lecture Notes in Computer Science, vol 3785. Springer, Berlin, Heidelberg
https://doi.org/10.1007/11576280_5
https://www.researchgate.net/publication/220744129_29

[GLW2017] Hubert Garavel, Frédéric Lang, and Wendelin Serwe "From LOTOS to LNT" in ModelEd, TestEd, TrustEd - Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday, volume 10500 of Lecture Notes in Computer Science, pages 3-26, October 2017 ftp://ftp.inrialpes.fr/pub/vasy/publications/cadp/Garavel-Lang-Serwe-17.pdf

[GMK2012] Grumberg O., Meller Y., Yorav K. (2012) “Applying Software Model Checking Techniques for Behavioral UML Models" In: Giannakopoulou D., Méry D. (eds) FM 2012: Formal Methods. FM 2012. Lecture Notes in Computer Science, vol 7436. Springer, Berlin, Heidelberg

[GRA] Graphviz - Graph Visualization Software, https://www.graphviz.org/

[HKLMPMS] Hvid Hansen H., Ketema J., Luttik B., Mousavi M., van de Pol J., dos Santos O.M. "Automated Verification of Executable UML Models". In: Aichernig B.K., de Boer F.S., Bonsangue M.M. (eds) Formal Methods for Components and Objects. FMCO 2010. Lecture Notes in Computer Science, vol 6957. Springer, Berlin, Heidelberg

[JDJLP] Toni Jussila1, Jori Dubrovin2, Tommi Junttila2, Timo Latvala3, and Ivan Porres4

"Model Checking Dynamic and Hierarchical UML State Machines" 3rd Workshop on Model Design and Validation (MoDeVa 2006), Genova, Italy, 2006

[KMR2020] Knapp A., Merz S., Rauh C. (2002) "Model Checking Timed UML State Machines and Collaborations" In: Damm W., Olderog E.R. (eds) Formal Techniques in Real-Time and Fault-Tolerant Systems. FTRTFT 2002. Lecture Notes in Computer Science, vol 2469. Springer, Berlin, Heidelberg

[L2013] Liu S. et al. (2013) "A Formal Semantics for Complete UML State Machines with Communications" In: Johnsen E.B., Petre L. (eds) Integrated Formal Methods. IFM 2013. Lecture Notes in Computer Science, vol 7940. Springer, Berlin, Heidelberg https://link.springer.com/chapter/10.1007/978-3-642-38613-8_23

[MAAP2015] Shift2Rail Multi-Annual Action Plan 2015 https://shift2rail.org/wp-content/uploads/2013/07/S2R-JU-GB_Decision-N-15-2015-MAAP.pdf

[MAAP2017] Shift2Rail Multi-Annual Action Plan –Executive View - Part A (2017) https://shift2rail.org/wp-content/uploads/2018/04/Maap_2018_FINAL_2.pdf

[MAAP2019] Shift2Rail Multi-Annual Action Plan – Part B (2019) https://shift2rail.org/wp-content/uploads/2019/05/Draft-Shift2Rail-Multi-Annual-Action-Plan-Part-B-20.5.2019.pdf

[MCRL2] mCRL2 website https://www.mcrl2.org/

[NPS2009] Artur Niewiadomski, Wojciech Penczek, Maciej Szreter "A New Approach to Model Checking of UML State Machines" Fundamenta Informaticae 93 (2009) 289–303 289 DOI 10.3233/FI-2009-103

[OD2017] Raquel Oliveira and Jürgen Dingel. "Supporting Model Refinement with Equivalence Checking in the Context of Model-Driven Engineering with UML-RT". Proceedings of the 14th Workshop on Model Engineering, Verification and Validation (MoDeVVa 2017), Austin, Texas, USA, pages 307-314, CEUR, September 2017. http://ceur-ws.org/Vol-2019/modevva_2.pdf

[OMG-SysML] Object Management Group, "SysML 1.6 Specification", November 2019. http://www.omg.org/spec/SysML/1.6/

[OMG-UML] Object Management Group "Unified Modelling Language" https://www.omg.org/spec/UML/About-UML/

[OMG-fUML1] OMG "Semantics of a Foundational Subset for Executable UML Models (fUML)" https://www.omg.org/spec/FUML/1.4

[OMG-fUML2]] Modeldriven, "The fUML Reference Implementation" https://github.com/ModelDriven/fUML-Reference-Implementation/blob/master/README.md

[OMG-Alf] OMG "Action Language for Foundational UML (Alf)" https://www.omg.org/spec/ALF/1.1

[OMG-Alf-Spec] OMG "Alf Specification" https://www.omg.org/spec/ALF/1.1/PDF

[OMG-PSCS] Object Management Group "Precise Semantics of UML Composite Structure (PSCS)" https://www.omg.org/spec/PSCS/1.2

[OR17] O'Regan G. (2017). "Concise Guide to Formal Methods - Theory, Fundamentals and Industry Applications", Undergraduate Topics in Computer Science, Springer.

[OSG2004] Ober I., Graf S., Ober I. "Validation of UML Models via a Mapping to Communicating Extended Timed Automata" In: Graf S., Mounier L. (eds) Model Checking

Software. SPIN 2004. Lecture Notes in Computer Science, vol 2989. Springer, Berlin, Heidelberg

[PEML] Jean-François Pétin, Dominique Evrot, Gérard Morel, Pascal Lamy "Combining SysML and formal models for safety requirements verification" https://hal.archives-ouvertes.fr/hal-00533311/document

[PROB] ProB website, https://www3.hhu.de/stups/prob/

[PTC-Windchill] PTC "Windchill Modeler SySim" https://www.ptc.com/en/products/plm/plm-products/windchill/modeler/sysim

[RBS2019] Abdul Rasheeq, Randolf Berglehner, and Colin Snook "Formal Specification of Railway Signalling System in SysML and UML-B", in RSSRail 2019 DOI: 10.13140/RG.2.2.21925.45288

[SG30] Anthony J.H. Simos, Ian Graham, "30 Things that go wrong in object modelling with UML 1.3." In: Behavioral Specifications of Businesses and Systems. The Springer International Series in Engineering and Computer Science, vol 523. Springer, Boston, MA https://doi.org/10.1007/978-1-4615-5229-1_17 http://staffwww.dcs.shef.ac.uk/people/A.Simons/research/papers/uml30things.pdf

[SPARX] SPARX Systems Enterprise Architect https://sparxsystems.com/products/ea/index.html

[SSB2012] Snook C., Savicks V., Butler M. (2011) "Verification of UML Models by Translation to UML-B". In: Aichernig B.K., de Boer F.S., Bonsangue M.M. (eds) Formal Methods for Components and Objects. FMCO 2010. Lecture Notes in Computer Science, vol 6957. Springer, Berlin, Heidelberg

[SUB-039] UNISIG - "FIS for the RBC/RBC Handover" - SUBSET-039 - 17-12-2015 (Issue 3.2.0)

[SUB-098] UNISIG - "RBC/RBC Safe Communication Interface" - SUBSET-098 - 21-05-2007

[TWC01] Tretmans, J., Wijbrans, K., Chaudron, M.R.W. (2001). "Software Engineering with Formal Methods: The Development of a Storm Surge Barrier Control System Revisiting Seven Myths of Formal Methods". Formal Methods in System Design, 19(2): 195-215.

[UIC] European Union Agency for Railways https://uic.org/

[UMC1] KandISTI project website http://fmt.isti.cnr.it/kandisti

[UMC2] UMC project website http://fmt.isti.cnr.it/umc

[UNISIG] UNISIG is an industrial consortium factsheet http://www.ertms.net/wp-content/uploads/2014/09/ERTMS_Factsheet_8_UNISIG.pdf

[UPPAAL] UPPAAL Web site http://www.uppaal.org/

[WHATISUML] Visual paradigm, "What is Unified Modeling Language (UML)"? https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

[WOD12] Woodcock, J., Larsen, P.G., Bicarregui, J., & Fitzgerald, J. (2009). "Formal methods: Practice and experience". ACM Computing Surveys, 41(4): 1-36..

[X2RAIL2] ASTRail project website https://projects.shift2rail.org/s2r_ip2_n.aspx?p=X2RAIL-2

[X2R2-D51] X2Rail project, Deliverable D5.1 "Formal Methods (Taxonomy and Survey), Proposed Methods and Applications" https://projects.shift2rail.org/download.aspx?id=b4cf6a3d-f1f2-4dd3-ae01-2bada34596b8

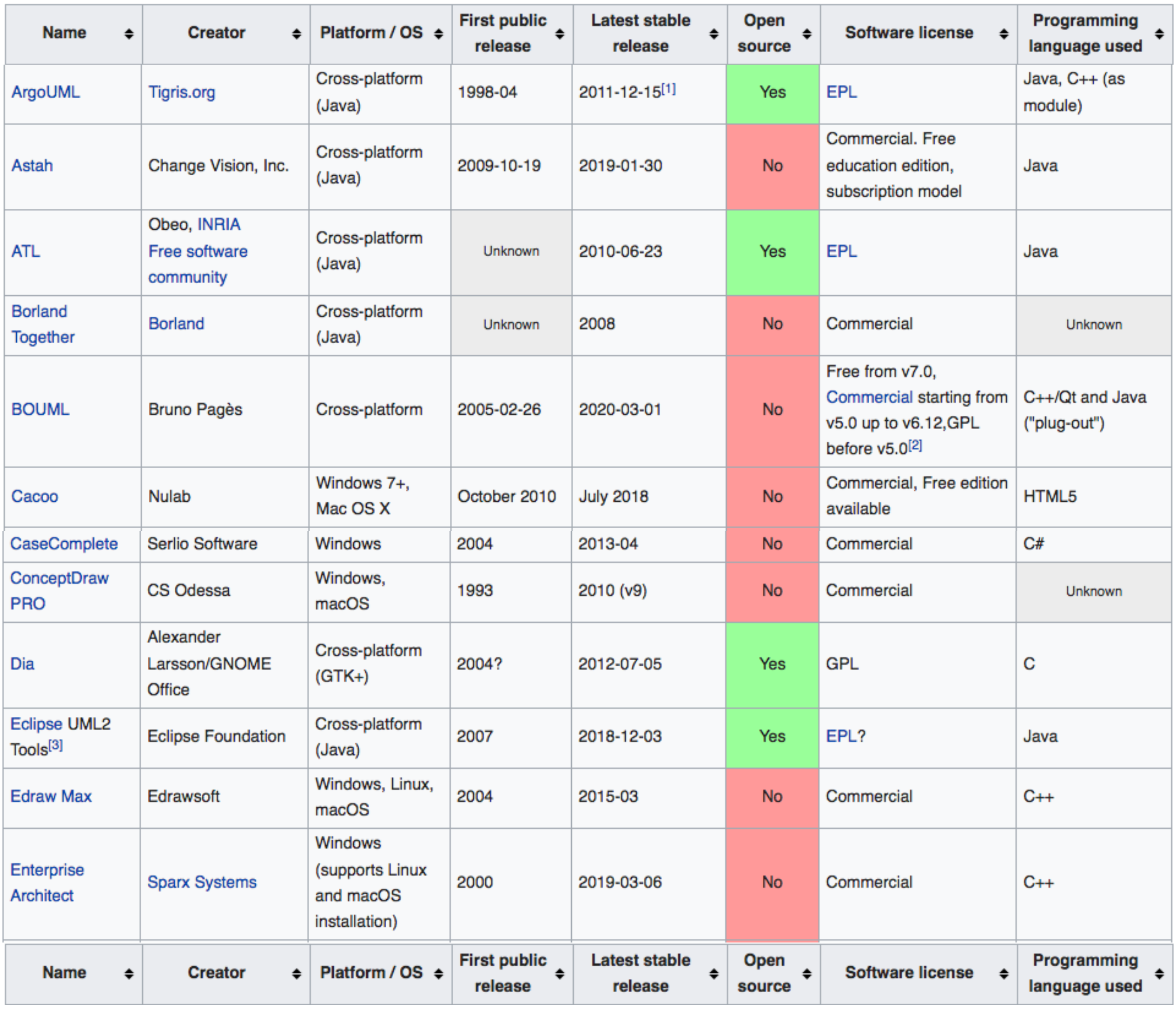
[Y2010] S. J. Zhang and Y. Liu, "An Automatic Approach to Model Checking UML State Machines,"2010 Fourth International Conference on Secure Software Integration and Reliability Improvement Companion, Singapore, 2010, pp. 1-6, doi: 10.1109/SSIRI-C.2010.11.

[YLWD] W. L. Yeung, K. R. P. H. Leung, Ji Wang and Wei Dong, "Improvements towards formalizing UML state diagrams in CSP," 12th Asia-Pacific Software Engineering Conference (APSEC'05), Taipei, Taiwan, 2005, pp. 7 pp.-, doi: 10.1109/APSEC.2005.70.

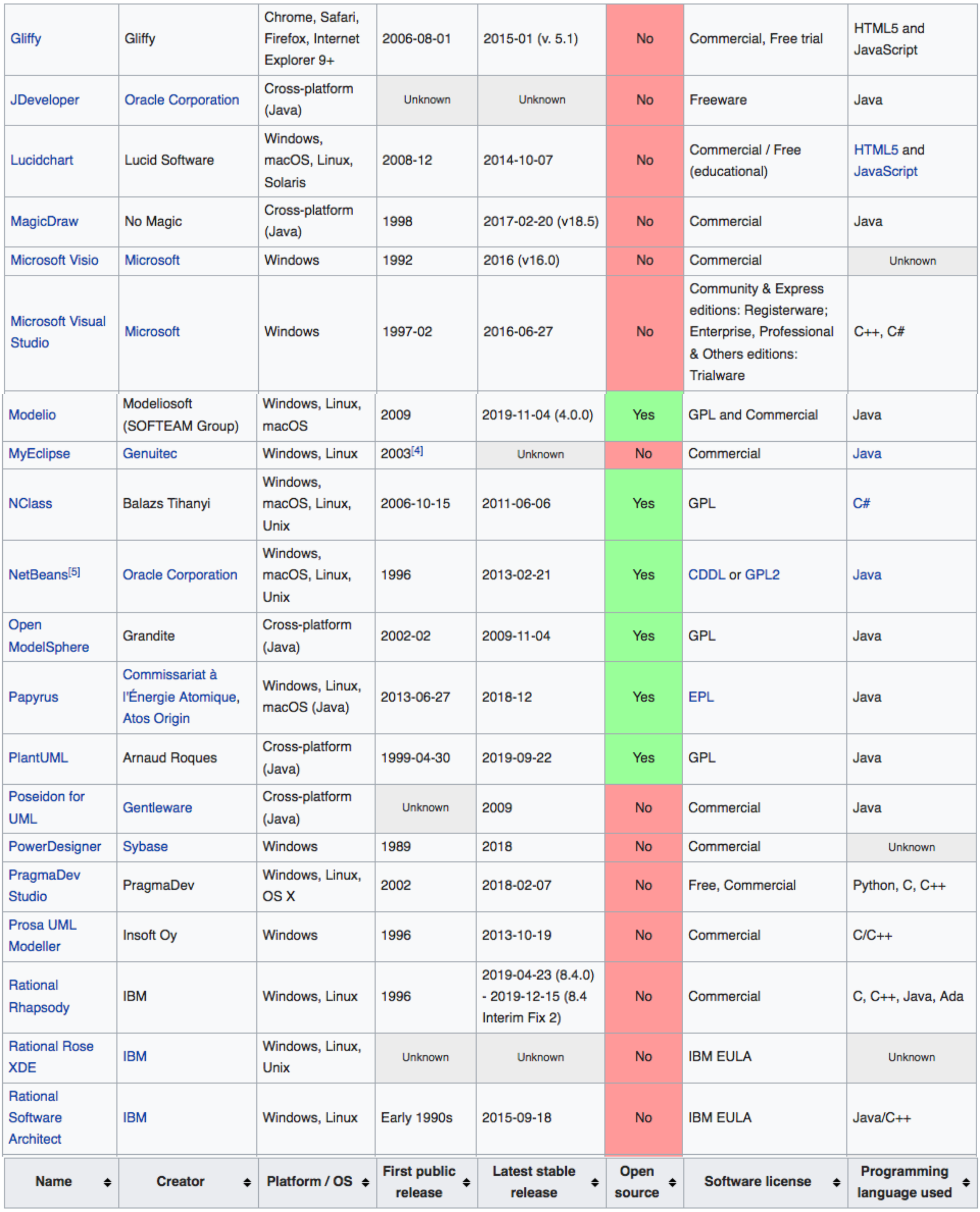
8 Informative Annexes

8.1 List of UML tools

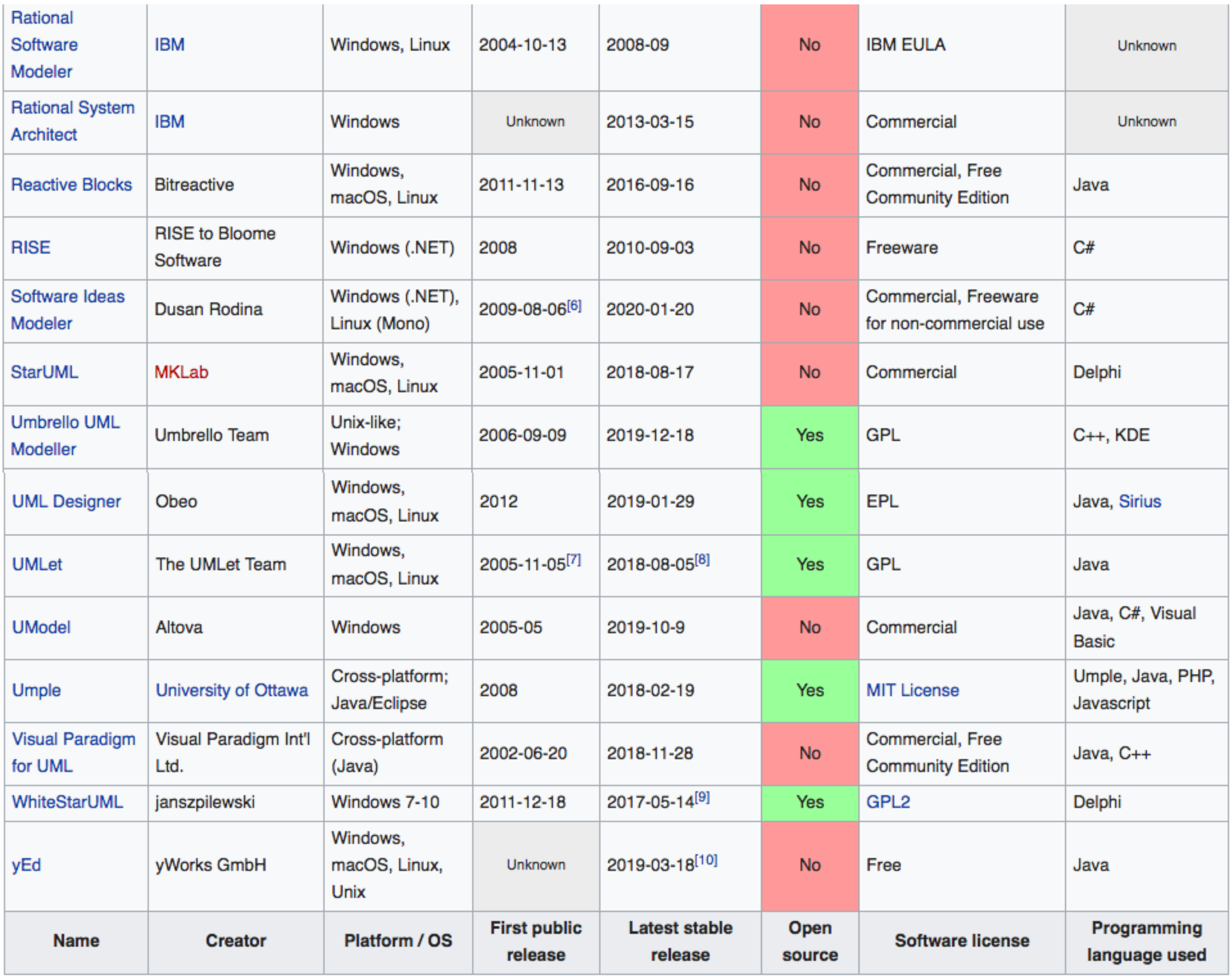
The following (not exhaustive) list of UML tools, as it appears on Wikipedia, (https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools), has just the purpose of giving an indication on how complex and heterogeneous is the ecosystem of UML tools. It is definitely out-of-scope for the project to make an overall survey on this aspect, or to identify which of these tools fits at best or demonstrator needs. This list, however, gives an overview of the high degree of freedom that is currently available.



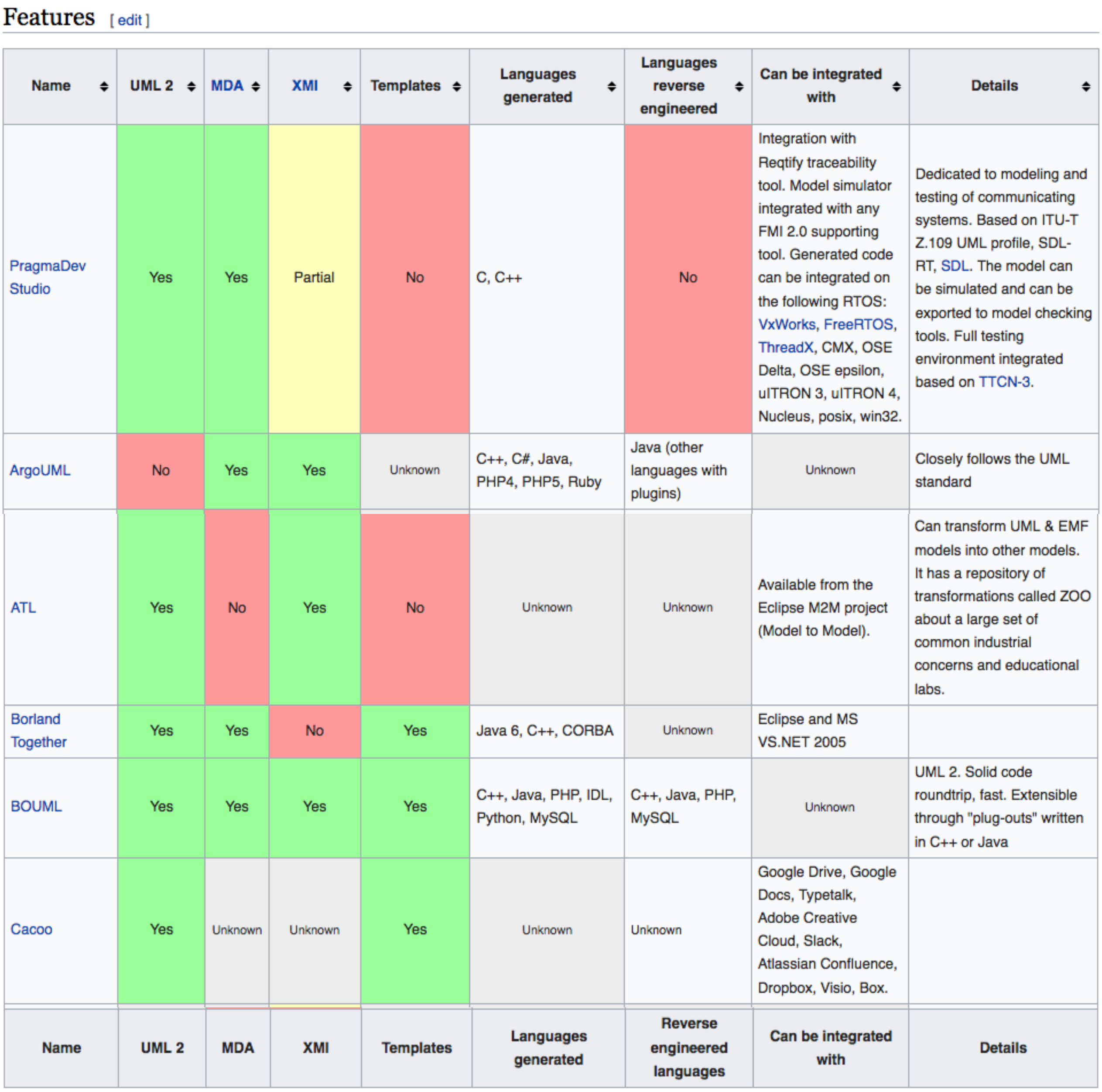
Name	Creator	Platform / OS	First public release	Latest stable release	Open source	Software license	Programming language used
ArgoUML	Tigris.org	Cross-platform (Java)	1998-04	2011-12-15[1]	Yes	EPL	Java, C++ (as module)
Astah	Change Vision, Inc.	Cross-platform (Java)	2009-10-19	2019-01-30	No	Commercial. Free education edition, subscription model	Java
ATL	Obeo, INRIA Free software community	Cross-platform (Java)	Unknown	2010-06-23	Yes	EPL	Java
Borland Together	Borland	Cross-platform (Java)	Unknown	2008	No	Commercial	Unknown
BOUML	Bruno Pagès	Cross-platform	2005-02-26	2020-03-01	No	Free from v7.0, Commercial starting from v5.0 up to v6.12,GPL before v5.0[2]	C++/Qt and Java ("plug-out")
Cacoo	Nulab	Windows 7+, Mac OS X	October 2010	July 2018	No	Commercial, Free edition available	HTML5
CaseComplete	Serlio Software	Windows	2004	2013-04	No	Commercial	C#
ConceptDraw PRO	CS Odessa	Windows, macOS	1993	2010 (v9)	No	Commercial	Unknown
Dia	Alexander Larsson/GNOME Office	Cross-platform (GTK+)	2004?	2012-07-05	Yes	GPL	C
Eclipse UML2 Tools[3]	Eclipse Foundation	Cross-platform (Java)	2007	2018-12-03	Yes	EPL?	Java
Edraw Max	Edrawsoft	Windows, Linux, macOS	2004	2015-03	No	Commercial	C++
Enterprise Architect	Sparx Systems	Windows (supports Linux and macOS installation)	2000	2019-03-06	No	Commercial	C++
Name	Creator	Platform / OS	First public release	Latest stable release	Open source	Software license	Programming language used



Gliffy	Gliffy	Chrome, Safari, Firefox, Internet Explorer 9+	2006-08-01	2015-01 (v. 5.1)	No	Commercial, Free trial	HTML5 and JavaScript
JDeveloper	Oracle Corporation	Cross-platform (Java)	Unknown	Unknown	No	Freeware	Java
Lucidchart	Lucid Software	Windows, macOS, Linux, Solaris	2008-12	2014-10-07	No	Commercial / Free (educational)	HTML5 and JavaScript
MagicDraw	No Magic	Cross-platform (Java)	1998	2017-02-20 (v18.5)	No	Commercial	Java
Microsoft Visio	Microsoft	Windows	1992	2016 (v16.0)	No	Commercial	Unknown
Microsoft Visual Studio	Microsoft	Windows	1997-02	2016-06-27	No	Community & Express editions: Registerware; Enterprise, Professional & Others editions: Trialware	C++, C#
Modelio	Modeliosoft (SOFTEAM Group)	Windows, Linux, macOS	2009	2019-11-04 (4.0.0)	Yes	GPL and Commercial	Java
MyEclipse	Genuitec	Windows, Linux	2003[4]	Unknown	No	Commercial	Java
NClass	Balazs Tihanyi	Windows, macOS, Linux, Unix	2006-10-15	2011-06-06	Yes	GPL	C#
NetBeans[5]	Oracle Corporation	Windows, macOS, Linux, Unix	1996	2013-02-21	Yes	CDDL or GPL2	Java
Open ModelSphere	Grandite	Cross-platform (Java)	2002-02	2009-11-04	Yes	GPL	Java
Papyrus	Commissariat à l'Énergie Atomique, Atos Origin	Windows, Linux, macOS (Java)	2013-06-27	2018-12	Yes	EPL	Java
PlantUML	Arnaud Roques	Cross-platform (Java)	1999-04-30	2019-09-22	Yes	GPL	Java
Poseidon for UML	Gentleware	Cross-platform (Java)	Unknown	2009	No	Commercial	Java
PowerDesigner	Sybase	Windows	1989	2018	No	Commercial	Unknown
PragmaDev Studio	PragmaDev	Windows, Linux, OS X	2002	2018-02-07	No	Free, Commercial	Python, C, C++
Prosa UML Modeller	Insoft Oy	Windows	1996	2013-10-19	No	Commercial	C/C++
Rational Rhapsody	IBM	Windows, Linux	1996	2019-04-23 (8.4.0) - 2019-12-15 (8.4 Interim Fix 2)	No	Commercial	C, C++, Java, Ada
Rational Rose XDE	IBM	Windows, Linux, Unix	Unknown	Unknown	No	IBM EULA	Unknown
Rational Software Architect	IBM	Windows, Linux	Early 1990s	2015-09-18	No	IBM EULA	Java/C++
Name	Creator	Platform / OS	First public release	Latest stable release	Open source	Software license	Programming language used

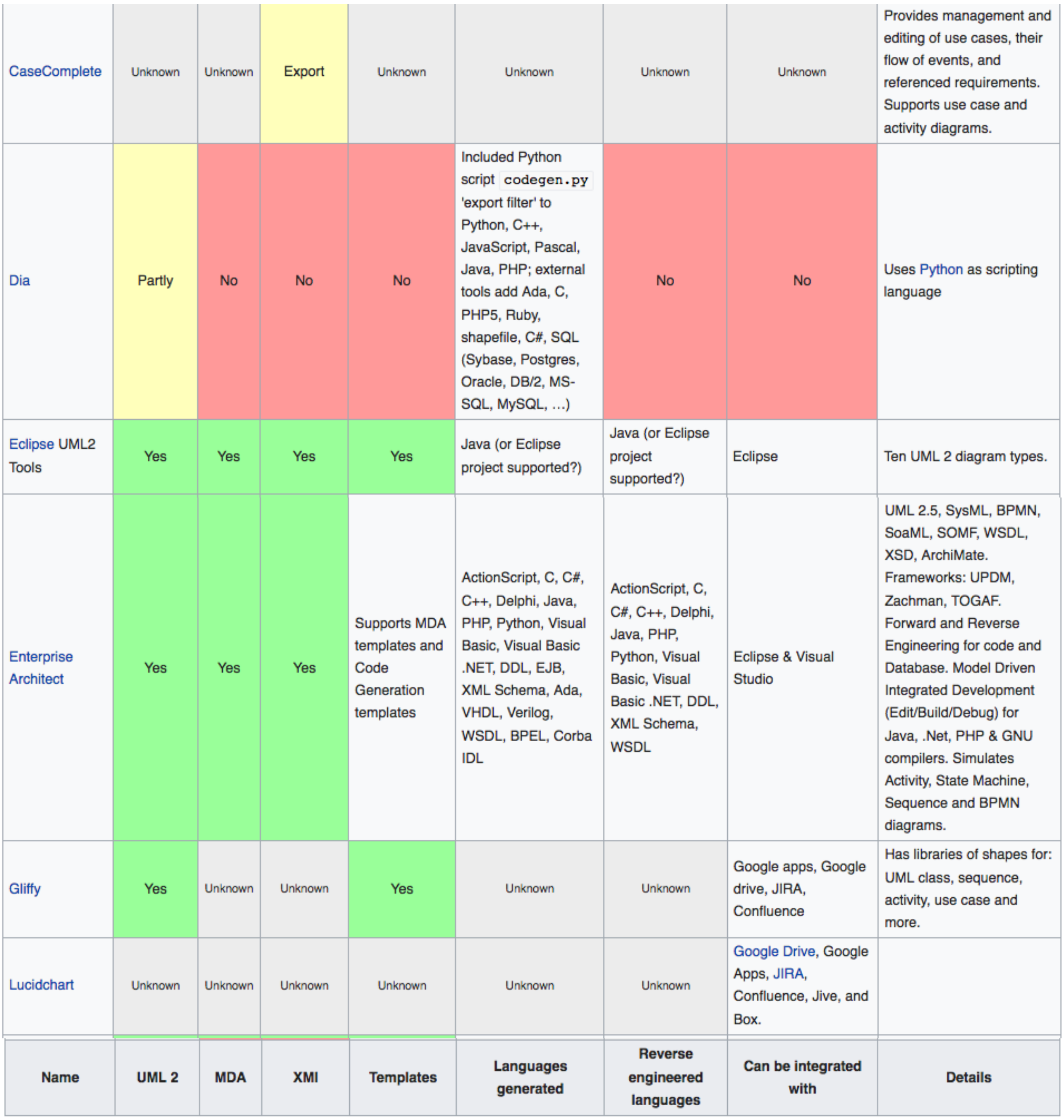


Rational Software Modeler	IBM	Windows, Linux	2004-10-13	2008-09	No	IBM EULA	Unknown
Rational System Architect	IBM	Windows	Unknown	2013-03-15	No	Commercial	Unknown
Reactive Blocks	Bitreactive	Windows, macOS, Linux	2011-11-13	2016-09-16	No	Commercial, Free Community Edition	Java
RISE	RISE to Bloome Software	Windows (.NET)	2008	2010-09-03	No	Freeware	C#
Software Ideas Modeler	Dusan Rodina	Windows (.NET), Linux (Mono)	2009-08-06[6]	2020-01-20	No	Commercial, Freeware for non-commercial use	C#
StarUML	MKLab	Windows, macOS, Linux	2005-11-01	2018-08-17	No	Commercial	Delphi
Umbrello UML Modeller	Umbrello Team	Unix-like; Windows	2006-09-09	2019-12-18	Yes	GPL	C++, KDE
UML Designer	Obeo	Windows, macOS, Linux	2012	2019-01-29	Yes	EPL	Java, Sirius
UMLet	The UMLet Team	Windows, macOS, Linux	2005-11-05[7]	2018-08-05[8]	Yes	GPL	Java
UModel	Altova	Windows	2005-05	2019-10-9	No	Commercial	Java, C#, Visual Basic
Umple	University of Ottawa	Cross-platform; Java/Eclipse	2008	2018-02-19	Yes	MIT License	Umple, Java, PHP, Javascript
Visual Paradigm for UML	Visual Paradigm Int'l Ltd.	Cross-platform (Java)	2002-06-20	2018-11-28	No	Commercial, Free Community Edition	Java, C++
WhiteStarUML	janszpilewski	Windows 7-10	2011-12-18	2017-05-14[9]	Yes	GPL2	Delphi
yEd	yWorks GmbH	Windows, macOS, Linux, Unix	Unknown	2019-03-18[10]	No	Free	Java
Name	**Creator**	**Platform / OS**	**First public release**	**Latest stable release**	**Open source**	**Software license**	**Programming language used**

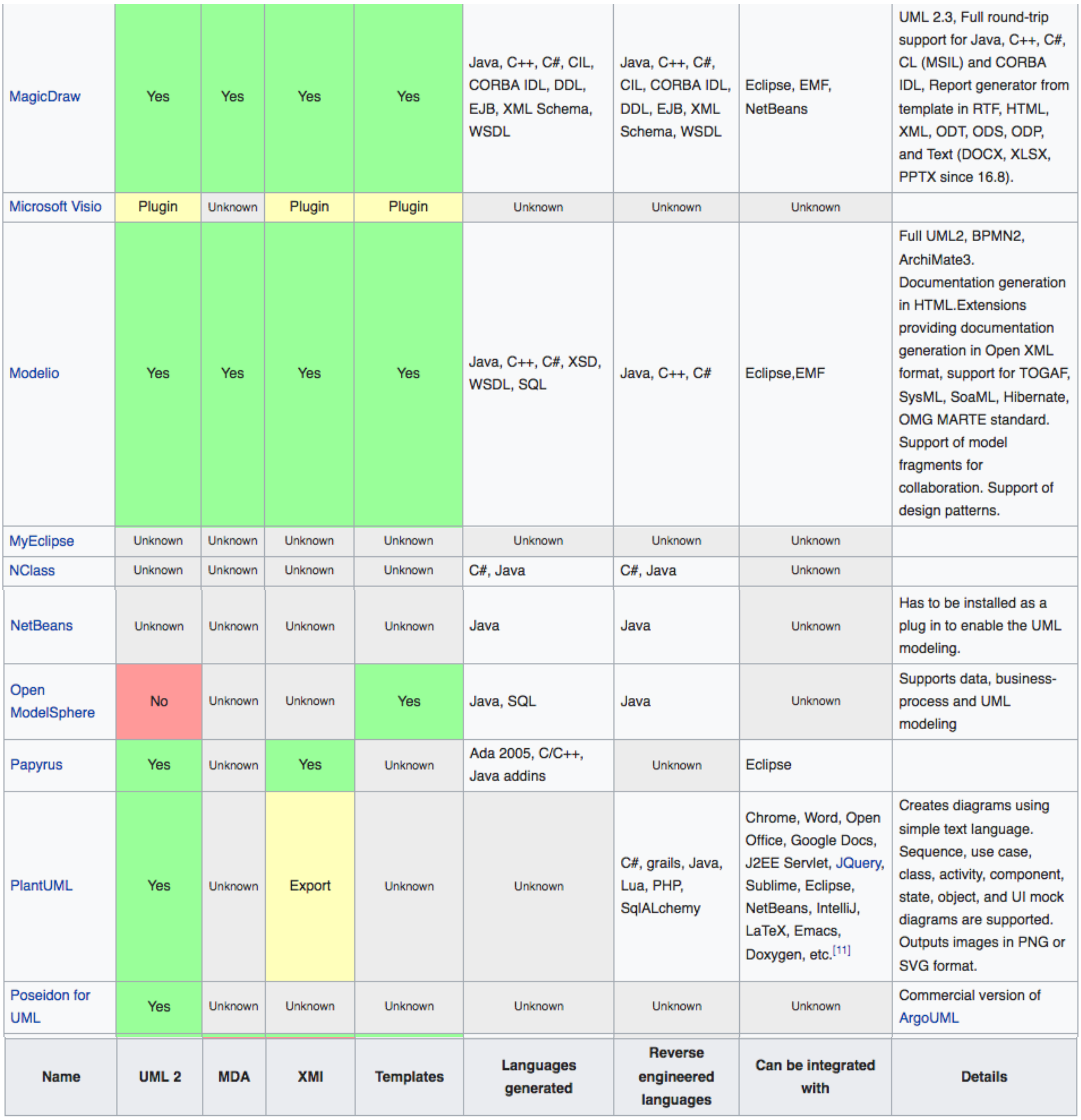


Features [edit]

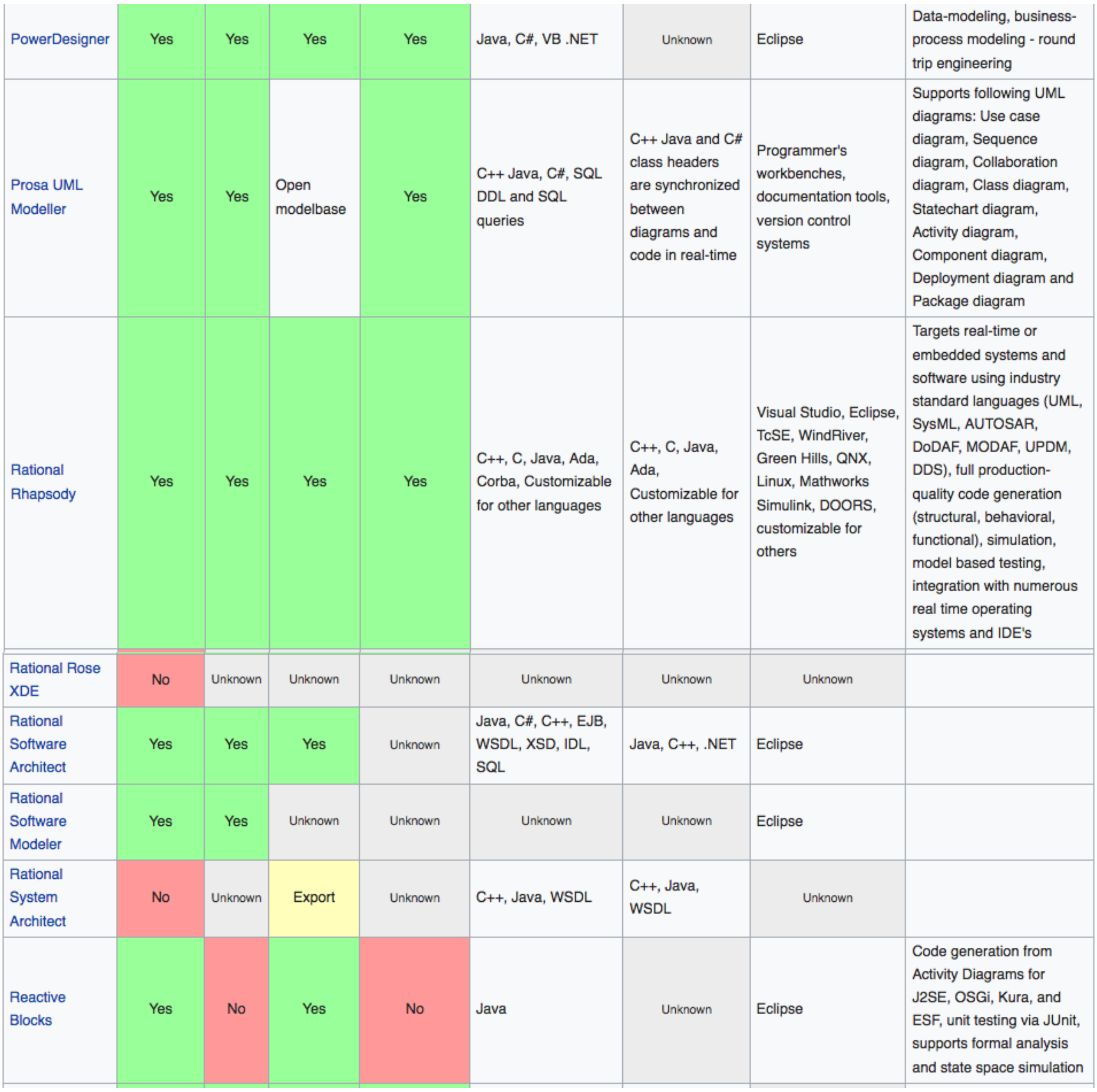
Name	UML 2	MDA	XMI	Templates	Languages generated	Languages reverse engineered	Can be integrated with	Details
PragmaDev Studio	Yes	Yes	Partial	No	C, C++	No	Integration with Reqtify traceability tool. Model simulator integrated with any FMI 2.0 supporting tool. Generated code can be integrated on the following RTOS: VxWorks, FreeRTOS, ThreadX, CMX, OSE Delta, OSE epsilon, uITRON 3, uITRON 4, Nucleus, posix, win32.	Dedicated to modeling and testing of communicating systems. Based on ITU-T Z.109 UML profile, SDL-RT, SDL. The model can be simulated and can be exported to model checking tools. Full testing environment integrated based on TTCN-3.
ArgoUML	No	Yes	Yes	Unknown	C++, C#, Java, PHP4, PHP5, Ruby	Java (other languages with plugins)	Unknown	Closely follows the UML standard
ATL	Yes	No	Yes	No	Unknown	Unknown	Available from the Eclipse M2M project (Model to Model).	Can transform UML & EMF models into other models. It has a repository of transformations called ZOO about a large set of common industrial concerns and educational labs.
Borland Together	Yes	Yes	No	Yes	Java 6, C++, CORBA	Unknown	Eclipse and MS VS.NET 2005	
BOUML	Yes	Yes	Yes	Yes	C++, Java, PHP, IDL, Python, MySQL	C++, Java, PHP, MySQL	Unknown	UML 2. Solid code roundtrip, fast. Extensible through "plug-outs" written in C++ or Java
Cacoo	Yes	Unknown	Unknown	Yes	Unknown	Unknown	Google Drive, Google Docs, Typetalk, Adobe Creative Cloud, Slack, Atlassian Confluence, Dropbox, Visio, Box.	
Name	UML 2	MDA	XMI	Templates	Languages generated	Reverse engineered languages	Can be integrated with	Details



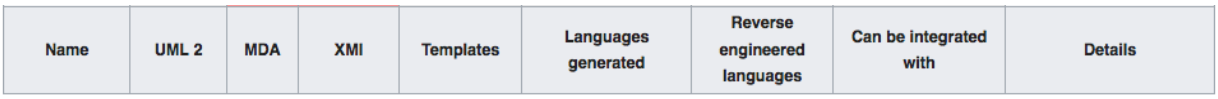
Name	UML 2	MDA	XMI	Templates	Languages generated	Reverse engineered languages	Can be integrated with	Details
CaseComplete	Unknown	Unknown	Export	Unknown	Unknown	Unknown	Unknown	Provides management and editing of use cases, their flow of events, and referenced requirements. Supports use case and activity diagrams.
Dia	Partly	No	No	No	Included Python script `codegen.py` 'export filter' to Python, C++, JavaScript, Pascal, Java, PHP; external tools add Ada, C, PHP5, Ruby, shapefile, C#, SQL (Sybase, Postgres, Oracle, DB/2, MS-SQL, MySQL, ...)	No	No	Uses Python as scripting language
Eclipse UML2 Tools	Yes	Yes	Yes	Yes	Java (or Eclipse project supported?)	Java (or Eclipse project supported?)	Eclipse	Ten UML 2 diagram types.
Enterprise Architect	Yes	Yes	Yes	Supports MDA templates and Code Generation templates	ActionScript, C, C#, C++, Delphi, Java, PHP, Python, Visual Basic, Visual Basic .NET, DDL, EJB, XML Schema, Ada, VHDL, Verilog, WSDL, BPEL, Corba IDL	ActionScript, C, C#, C++, Delphi, Java, PHP, Python, Visual Basic, Visual Basic .NET, DDL, XML Schema, WSDL	Eclipse & Visual Studio	UML 2.5, SysML, BPMN, SoaML, SOMF, WSDL, XSD, ArchiMate. Frameworks: UPDM, Zachman, TOGAF. Forward and Reverse Engineering for code and Database. Model Driven Integrated Development (Edit/Build/Debug) for Java, .Net, PHP & GNU compilers. Simulates Activity, State Machine, Sequence and BPMN diagrams.
Gliffy	Yes	Unknown	Unknown	Yes	Unknown	Unknown	Google apps, Google drive, JIRA, Confluence	Has libraries of shapes for: UML class, sequence, activity, use case and more.
Lucidchart	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Google Drive, Google Apps, JIRA, Confluence, Jive, and Box.	



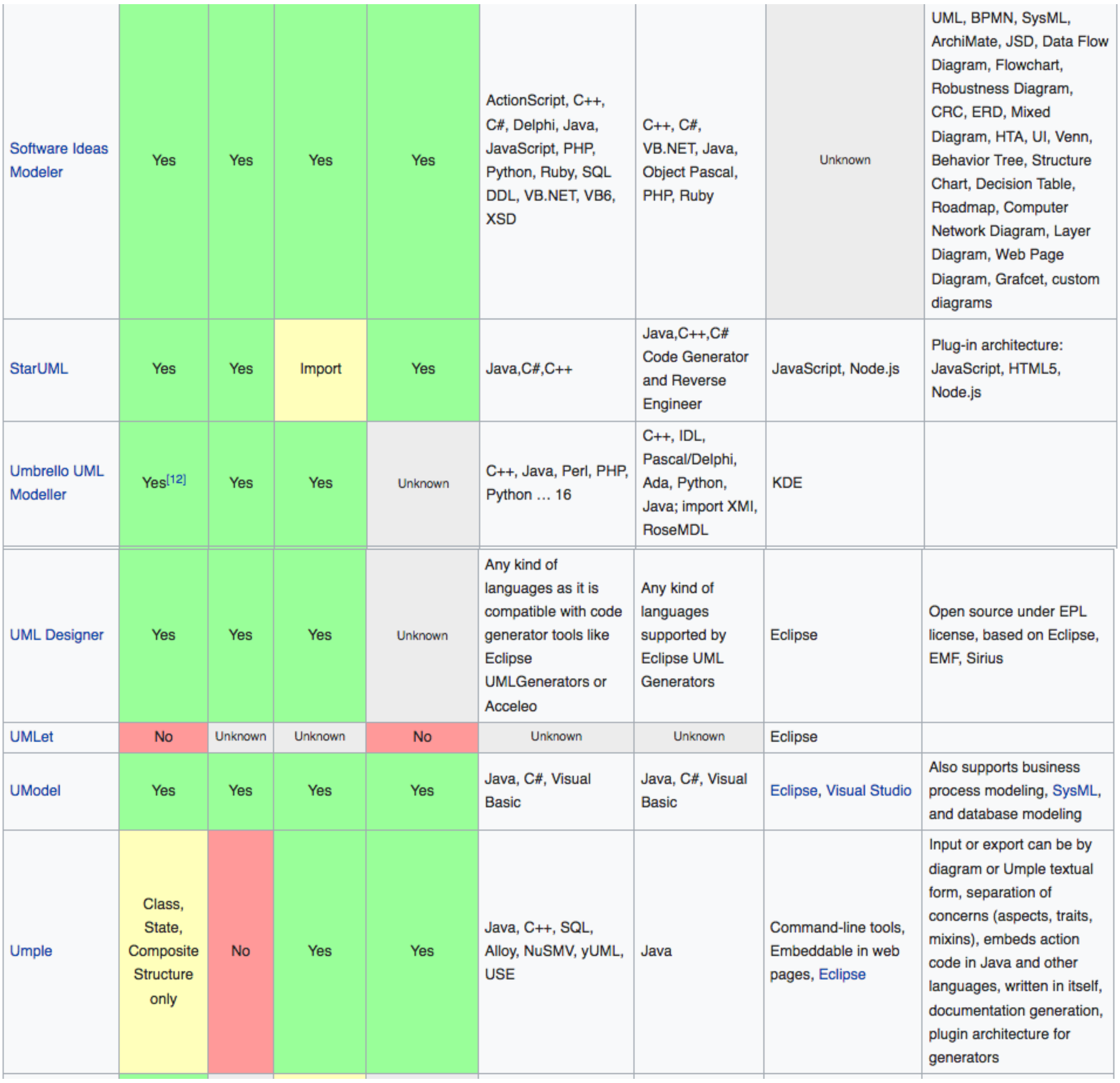
Name	UML 2	MDA	XMI	Templates	Languages generated	Reverse engineered languages	Can be integrated with	Details
MagicDraw	Yes	Yes	Yes	Yes	Java, C++, C#, CIL, CORBA IDL, DDL, EJB, XML Schema, WSDL	Java, C++, C#, CIL, CORBA IDL, DDL, EJB, XML Schema, WSDL	Eclipse, EMF, NetBeans	UML 2.3, Full round-trip support for Java, C++, C#, CL (MSIL) and CORBA IDL, Report generator from template in RTF, HTML, XML, ODT, ODS, ODP, and Text (DOCX, XLSX, PPTX since 16.8).
Microsoft Visio	Plugin	Unknown	Plugin	Plugin	Unknown	Unknown	Unknown	
Modelio	Yes	Yes	Yes	Yes	Java, C++, C#, XSD, WSDL, SQL	Java, C++, C#	Eclipse,EMF	Full UML2, BPMN2, ArchiMate3. Documentation generation in HTML.Extensions providing documentation generation in Open XML format, support for TOGAF, SysML, SoaML, Hibernate, OMG MARTE standard. Support of model fragments for collaboration. Support of design patterns.
MyEclipse	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	
NClass	Unknown	Unknown	Unknown	Unknown	C#, Java	C#, Java	Unknown	
NetBeans	Unknown	Unknown	Unknown	Unknown	Java	Java	Unknown	Has to be installed as a plug in to enable the UML modeling.
Open ModelSphere	No	Unknown	Unknown	Yes	Java, SQL	Java	Unknown	Supports data, business-process and UML modeling
Papyrus	Yes	Unknown	Yes	Unknown	Ada 2005, C/C++, Java addins	Unknown	Eclipse	
PlantUML	Yes	Unknown	Export	Unknown	Unknown	C#, grails, Java, Lua, PHP, SqlALchemy	Chrome, Word, Open Office, Google Docs, J2EE Servlet, JQuery, Sublime, Eclipse, NetBeans, IntelliJ, LaTeX, Emacs, Doxygen, etc.[11]	Creates diagrams using simple text language. Sequence, use case, class, activity, component, state, object, and UI mock diagrams are supported. Outputs images in PNG or SVG format.
Poseidon for UML	Yes	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Commercial version of ArgoUML



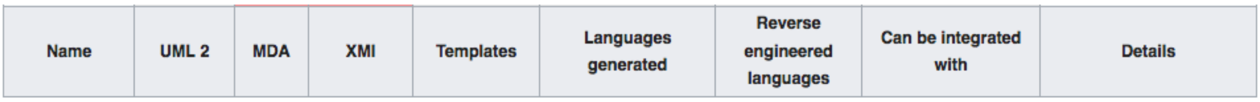
PowerDesigner	Yes	Yes	Yes	Yes	Java, C#, VB .NET	Unknown	Eclipse	Data-modeling, business-process modeling - round trip engineering
Prosa UML Modeller	Yes	Yes	Open modelbase	Yes	C++ Java, C#, SQL DDL and SQL queries	C++ Java and C# class headers are synchronized between diagrams and code in real-time	Programmer's workbenches, documentation tools, version control systems	Supports following UML diagrams: Use case diagram, Sequence diagram, Collaboration diagram, Class diagram, Statechart diagram, Activity diagram, Component diagram, Deployment diagram and Package diagram
Rational Rhapsody	Yes	Yes	Yes	Yes	C++, C, Java, Ada, Corba, Customizable for other languages	C++, C, Java, Ada, Customizable for other languages	Visual Studio, Eclipse, TcSE, WindRiver, Green Hills, QNX, Linux, Mathworks Simulink, DOORS, customizable for others	Targets real-time or embedded systems and software using industry standard languages (UML, SysML, AUTOSAR, DoDAF, MODAF, UPDM, DDS), full production-quality code generation (structural, behavioral, functional), simulation, model based testing, integration with numerous real time operating systems and IDE's
Rational Rose XDE	No	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	
Rational Software Architect	Yes	Yes	Yes	Unknown	Java, C#, C++, EJB, WSDL, XSD, IDL, SQL	Java, C++, .NET	Eclipse	
Rational Software Modeler	Yes	Yes	Unknown	Unknown	Unknown	Unknown	Eclipse	
Rational System Architect	No	Unknown	Export	Unknown	C++, Java, WSDL	C++, Java, WSDL	Unknown	
Reactive Blocks	Yes	No	Yes	No	Java	Unknown	Eclipse	Code generation from Activity Diagrams for J2SE, OSGi, Kura, and ESF, unit testing via JUnit, supports formal analysis and state space simulation



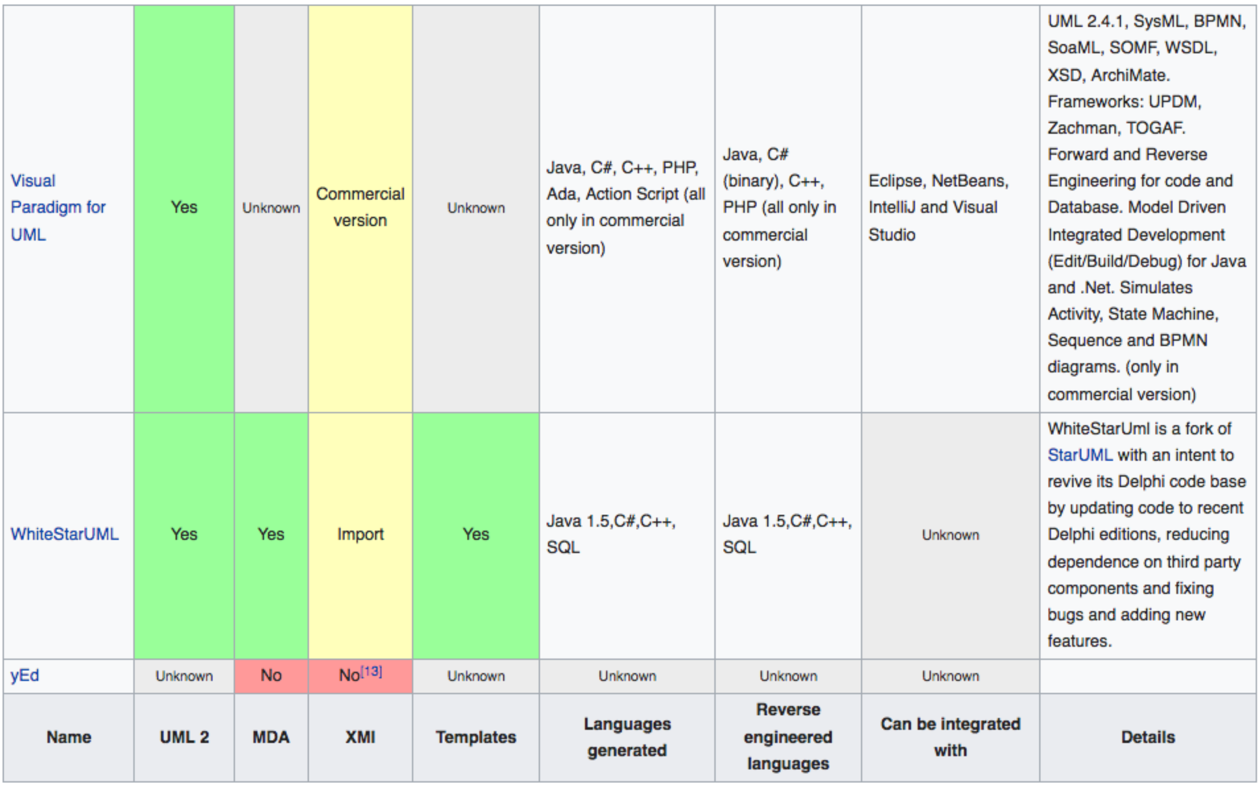
Name	UML 2	MDA	XMI	Templates	Languages generated	Reverse engineered languages	Can be integrated with	Details



Software Ideas Modeler	Yes	Yes	Yes	Yes	ActionScript, C++, C#, Delphi, Java, JavaScript, PHP, Python, Ruby, SQL DDL, VB.NET, VB6, XSD	C++, C#, VB.NET, Java, Object Pascal, PHP, Ruby	Unknown	UML, BPMN, SysML, ArchiMate, JSD, Data Flow Diagram, Flowchart, Robustness Diagram, CRC, ERD, Mixed Diagram, HTA, UI, Venn, Behavior Tree, Structure Chart, Decision Table, Roadmap, Computer Network Diagram, Layer Diagram, Web Page Diagram, Grafcet, custom diagrams
StarUML	Yes	Yes	Import	Yes	Java,C#,C++	Java,C++,C# Code Generator and Reverse Engineer	JavaScript, Node.js	Plug-in architecture: JavaScript, HTML5, Node.js
Umbrello UML Modeller	Yes[12]	Yes	Yes	Unknown	C++, Java, Perl, PHP, Python … 16	C++, IDL, Pascal/Delphi, Ada, Python, Java; import XMI, RoseMDL	KDE	
UML Designer	Yes	Yes	Yes	Unknown	Any kind of languages as it is compatible with code generator tools like Eclipse UMLGenerators or Acceleo	Any kind of languages supported by Eclipse UML Generators	Eclipse	Open source under EPL license, based on Eclipse, EMF, Sirius
UMLet	No	Unknown	Unknown	No	Unknown	Unknown	Eclipse	
UModel	Yes	Yes	Yes	Yes	Java, C#, Visual Basic	Java, C#, Visual Basic	Eclipse, Visual Studio	Also supports business process modeling, SysML, and database modeling
Umple	Class, State, Composite Structure only	No	Yes	Yes	Java, C++, SQL, Alloy, NuSMV, yUML, USE	Java	Command-line tools, Embeddable in web pages, Eclipse	Input or export can be by diagram or Umple textual form, separation of concerns (aspects, traits, mixins), embeds action code in Java and other languages, written in itself, documentation generation, plugin architecture for generators

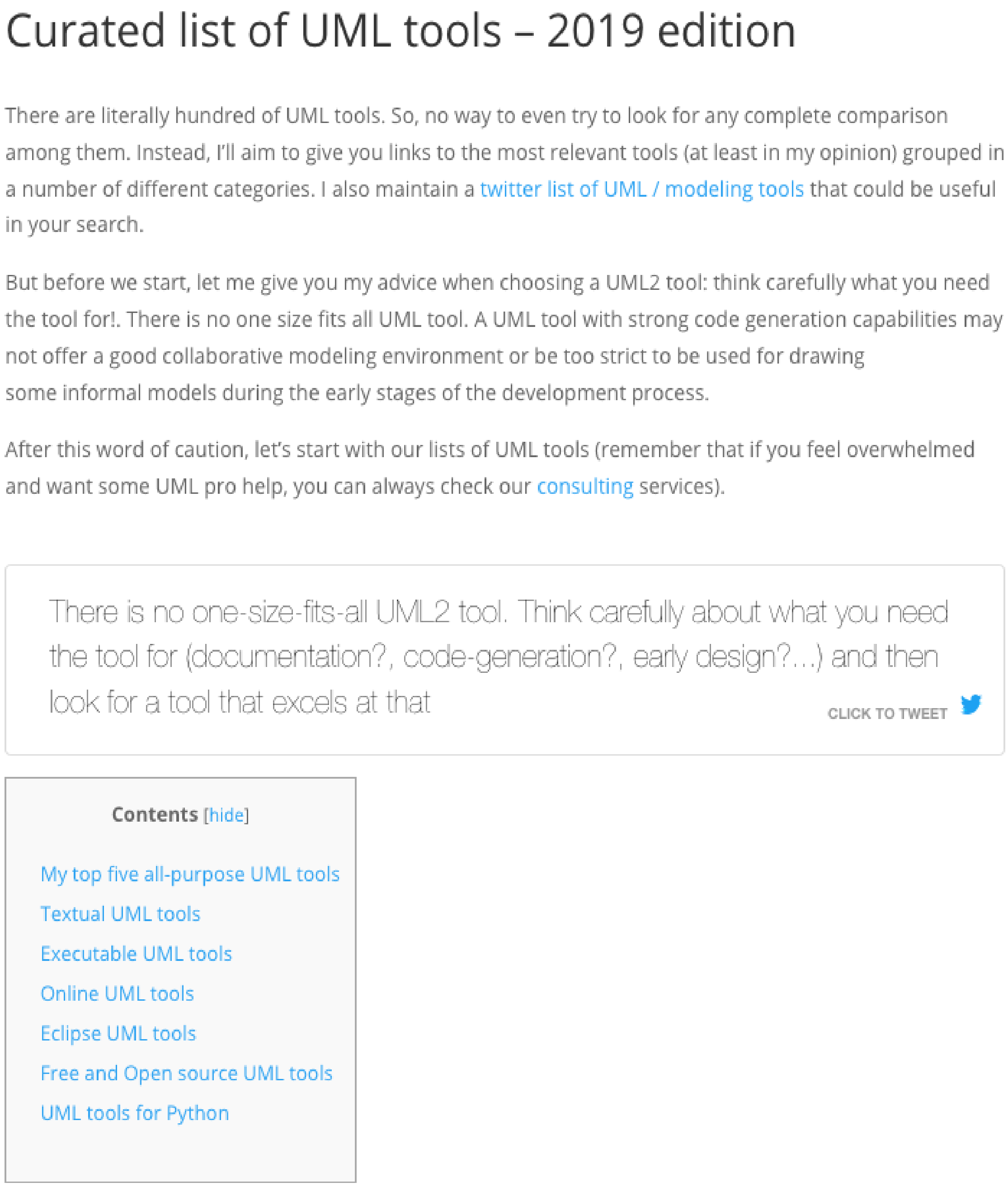


Name	UML 2	MDA	XMI	Templates	Languages generated	Reverse engineered languages	Can be integrated with	Details



Visual Paradigm for UML	Yes	Unknown	Commercial version	Unknown	Java, C#, C++, PHP, Ada, Action Script (all only in commercial version)	Java, C# (binary), C++, PHP (all only in commercial version)	Eclipse, NetBeans, IntelliJ and Visual Studio	UML 2.4.1, SysML, BPMN, SoaML, SOMF, WSDL, XSD, ArchiMate. Frameworks: UPDM, Zachman, TOGAF. Forward and Reverse Engineering for code and Database. Model Driven Integrated Development (Edit/Build/Debug) for Java and .Net. Simulates Activity, State Machine, Sequence and BPMN diagrams. (only in commercial version)
WhiteStarUML	Yes	Yes	Import	Yes	Java 1.5,C#,C++, SQL	Java 1.5,C#,C++, SQL	Unknown	WhiteStarUml is a fork of StarUML with an intent to revive its Delphi code base by updating code to recent Delphi editions, reducing dependence on third party components and fixing bugs and adding new features.
yEd	Unknown	No	No[13]	Unknown	Unknown	Unknown	Unknown	
Name	**UML 2**	**MDA**	**XMI**	**Templates**	**Languages generated**	**Reverse engineered languages**	**Can be integrated with**	**Details**

The site “Modelling Languages” (https://modeling-languages.com) describes in a well structured way the available resources and studies related to Modelling Languages. One of these resources is a “curated list of UML tools” (https://modeling-languages.com/text-uml-tools-complete-list/). The category “Textual UML tools” and “Executable UML tools” are of particular interest for our purposes.



Curated list of UML tools – 2019 edition

There are literally hundred of UML tools. So, no way to even try to look for any complete comparison among them. Instead, I’ll aim to give you links to the most relevant tools (at least in my opinion) grouped in a number of different categories. I also maintain a twitter list of UML / modeling tools that could be useful in your search.

But before we start, let me give you my advice when choosing a UML2 tool: think carefully what you need the tool for!. There is no one size fits all UML tool. A UML tool with strong code generation capabilities may not offer a good collaborative modeling environment or be too strict to be used for drawing some informal models during the early stages of the development process.

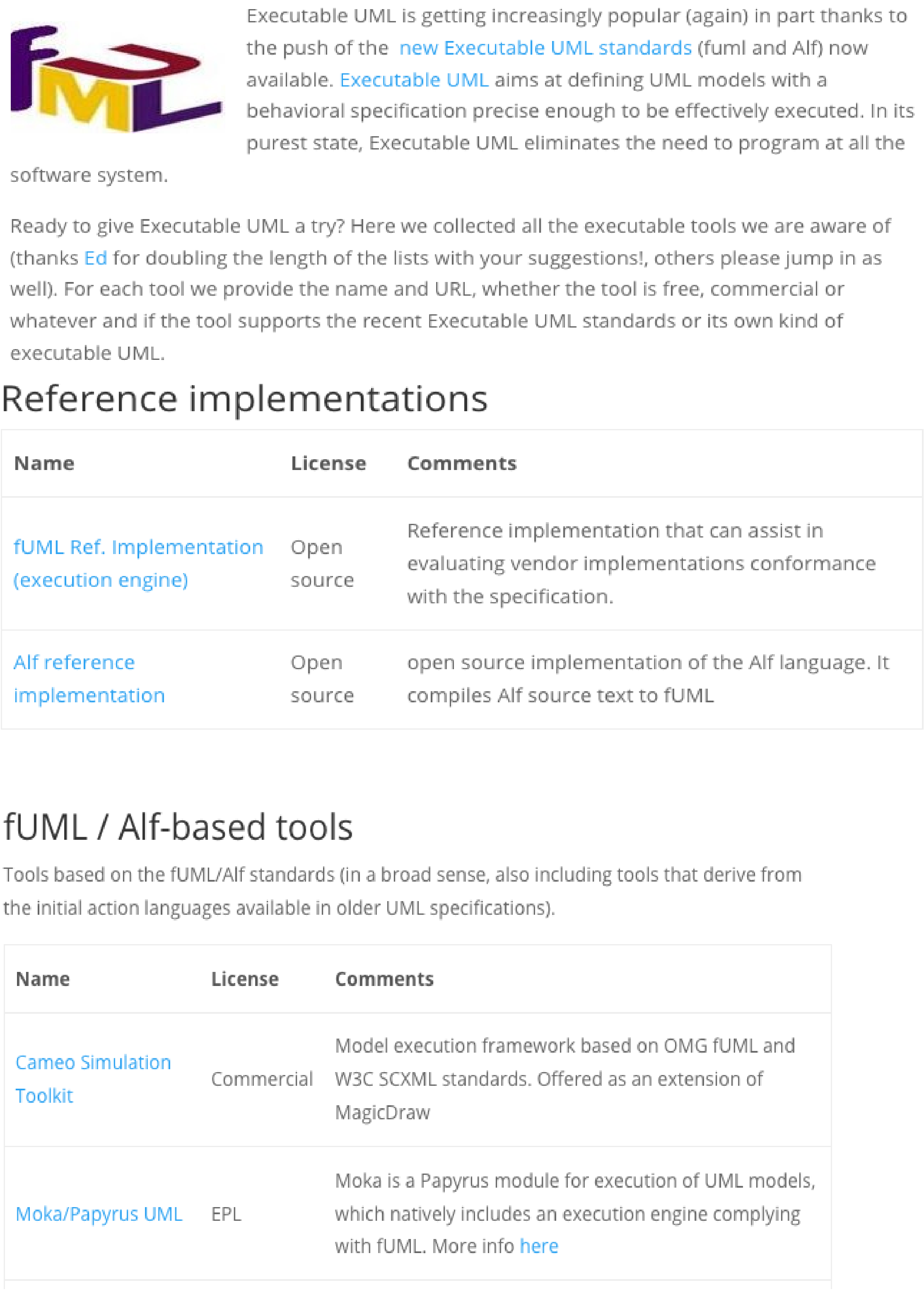
After this word of caution, let’s start with our lists of UML tools (remember that if you feel overwhelmed and want some UML pro help, you can always check our consulting services).

> There is no one-size-fits-all UML2 tool. Think carefully about what you need the tool for (documentation?, code-generation?, early design?…) and then look for a tool that excels at that

CLICK TO TWEET

Contents [hide]

from: https://modeling-languages.com/list-of-executable-uml-tools/



Executable UML is getting increasingly popular (again) in part thanks to the push of the new Executable UML standards (fuml and Alf) now available. Executable UML aims at defining UML models with a behavioral specification precise enough to be effectively executed. In its purest state, Executable UML eliminates the need to program at all the software system.

Ready to give Executable UML a try? Here we collected all the executable tools we are aware of (thanks Ed for doubling the length of the lists with your suggestions!, others please jump in as well). For each tool we provide the name and URL, whether the tool is free, commercial or whatever and if the tool supports the recent Executable UML standards or its own kind of executable UML.

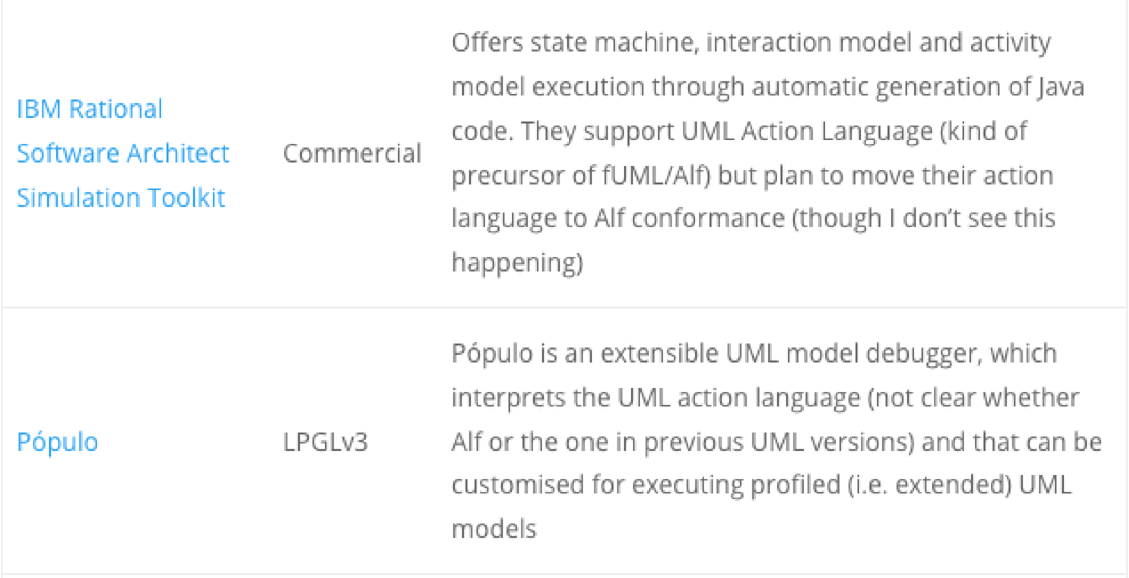
Reference implementations

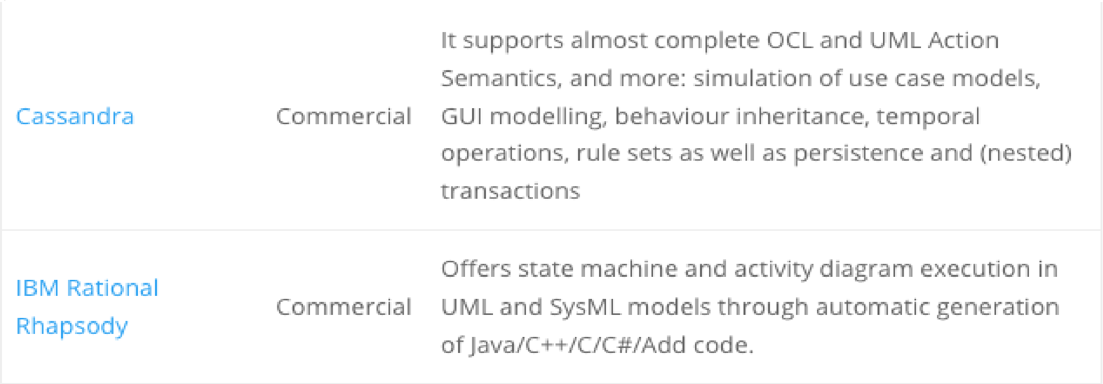
Name	License	Comments
fUML Ref. Implementation (execution engine)	Open source	Reference implementation that can assist in evaluating vendor implementations conformance with the specification.
Alf reference implementation	Open source	open source implementation of the Alf language. It compiles Alf source text to fUML

fUML / Alf-based tools

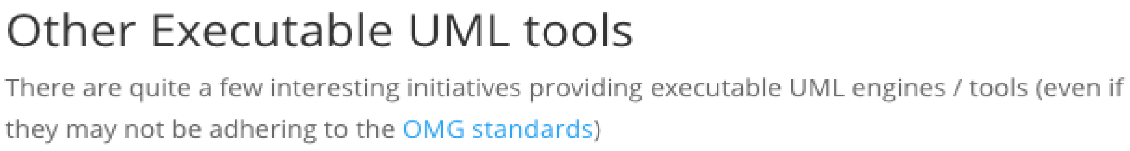
Tools based on the fUML/Alf standards (in a broad sense, also including tools that derive from the initial action languages available in older UML specifications).

Name	License	Comments
Cameo Simulation Toolkit	Commercial	Model execution framework based on OMG fUML and W3C SCXML standards. Offered as an extension of MagicDraw
Moka/Papyrus UML	EPL	Moka is a Papyrus module for execution of UML models, which natively includes an execution engine complying with fUML. More info here



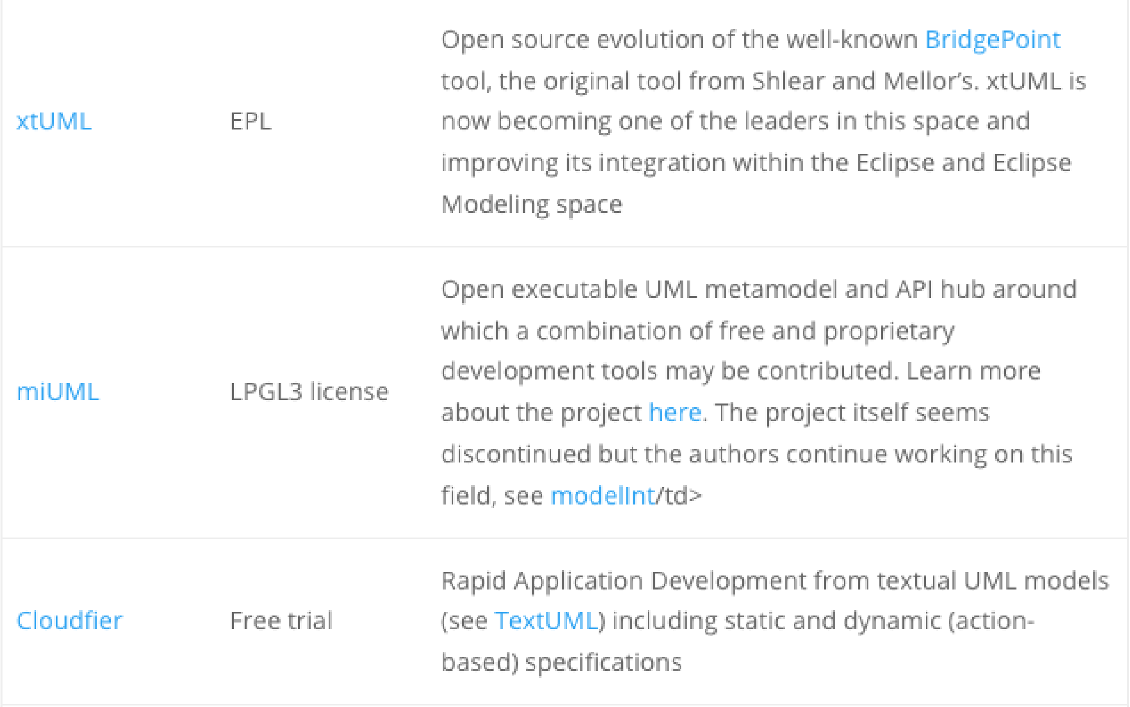


IBM Rational Software Architect Simulation Toolkit	Commercial	Offers state machine, interaction model and activity model execution through automatic generation of Java code. They support UML Action Language (kind of precursor of fUML/Alf) but plan to move their action language to Alf conformance (though I don't see this happening)
Pópulo	LPGLv3	Pópulo is an extensible UML model debugger, which interprets the UML action language (not clear whether Alf or the one in previous UML versions) and that can be customised for executing profiled (i.e. extended) UML models
Cassandra	Commercial	It supports almost complete OCL and UML Action Semantics, and more: simulation of use case models, GUI modelling, behaviour inheritance, temporal operations, rule sets as well as persistence and (nested) transactions
IBM Rational Rhapsody	Commercial	Offers state machine and activity diagram execution in UML and SysML models through automatic generation of Java/C++/C/C#/Add code.

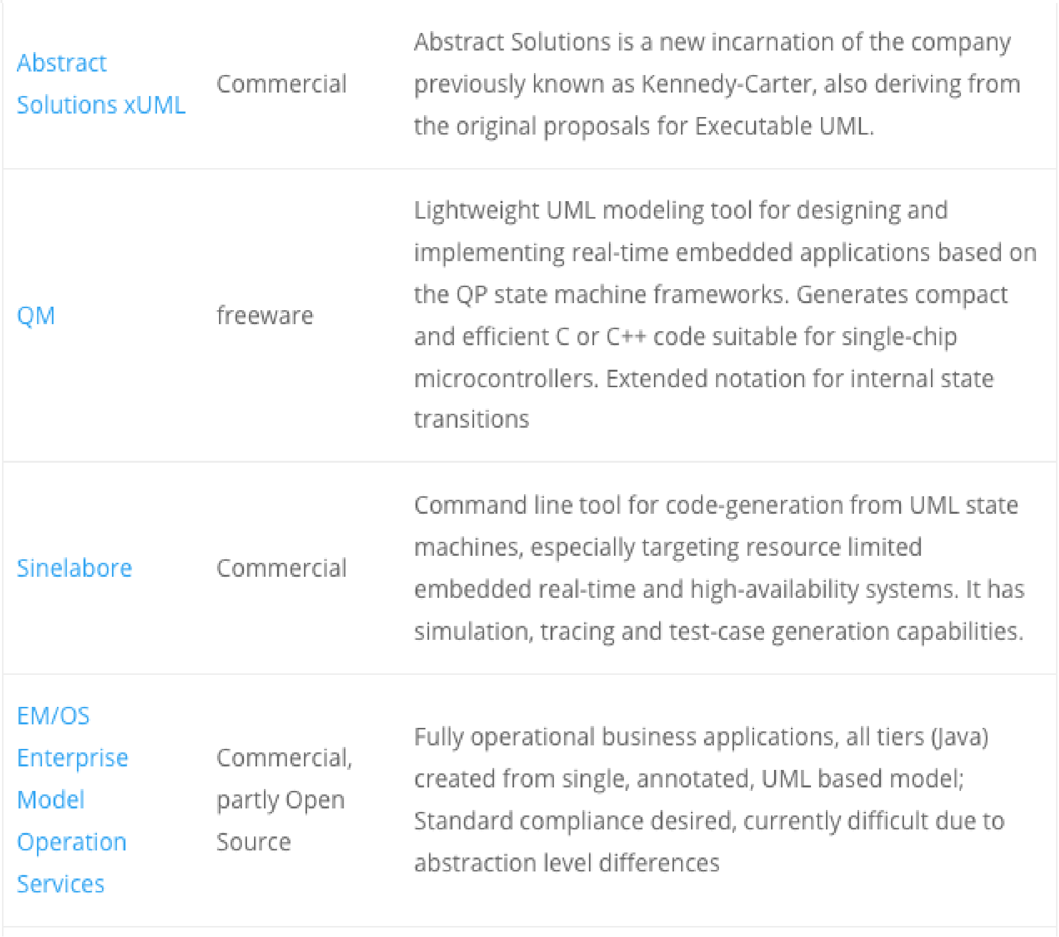


Other Executable UML tools

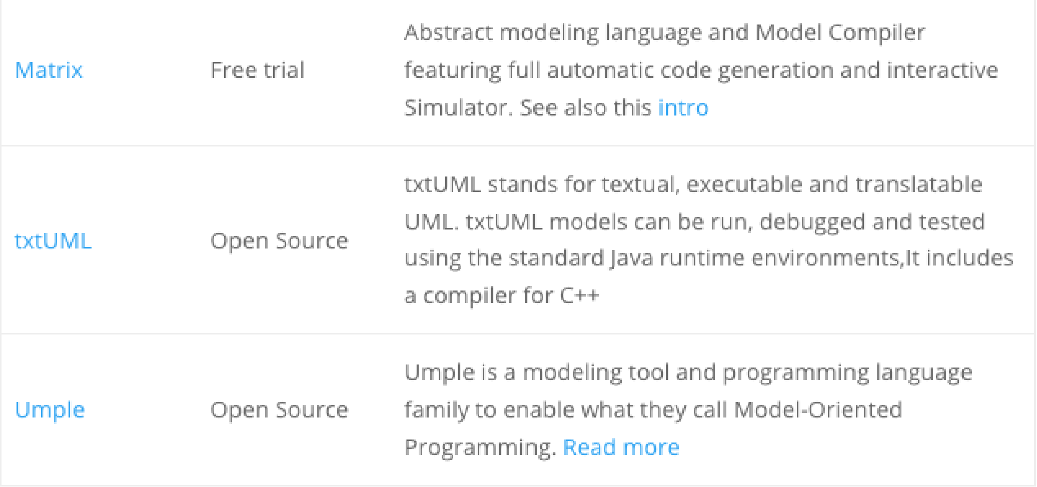
There are quite a few interesting initiatives providing executable UML engines / tools (even if they may not be adhering to the OMG standards)



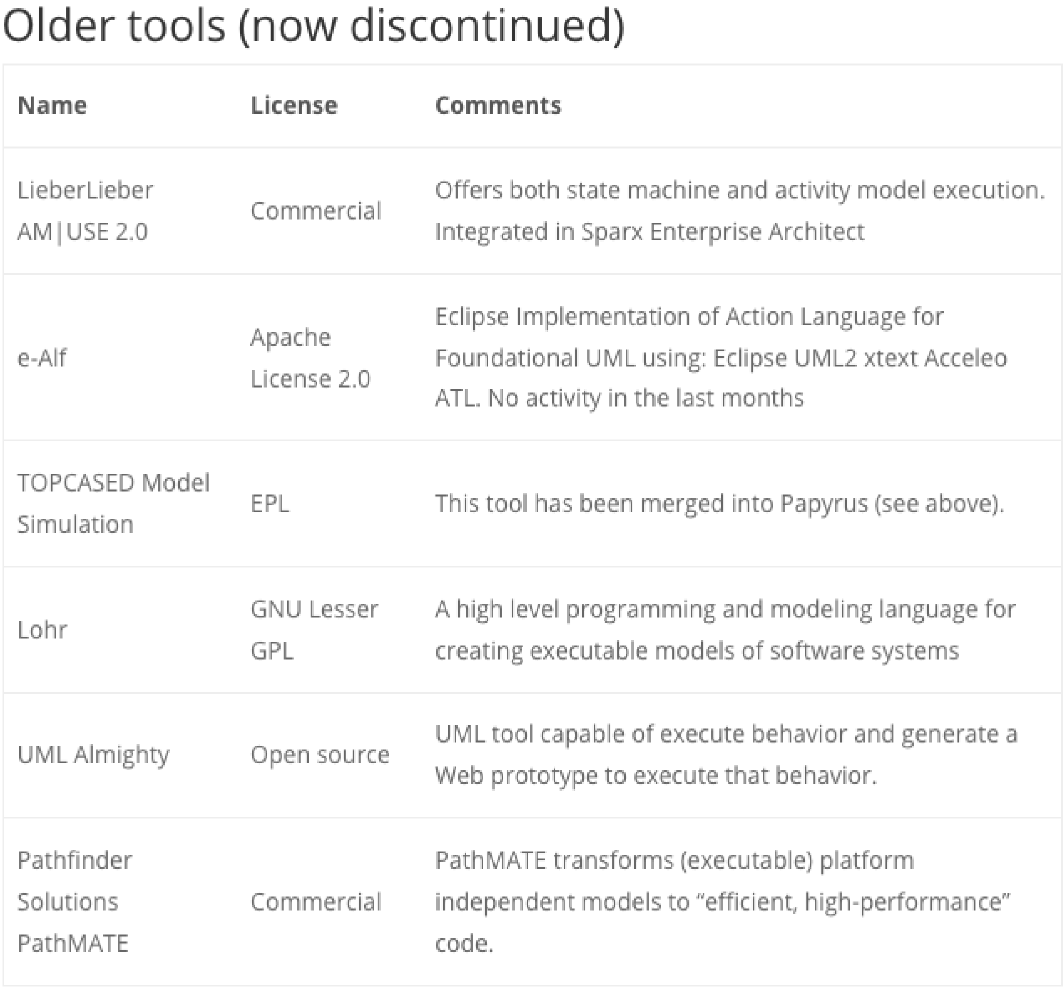
xtUML	EPL	Open source evolution of the well-known BridgePoint tool, the original tool from Shlear and Mellor's. xtUML is now becoming one of the leaders in this space and improving its integration within the Eclipse and Eclipse Modeling space
miUML	LPGL3 license	Open executable UML metamodel and API hub around which a combination of free and proprietary development tools may be contributed. Learn more about the project here. The project itself seems discontinued but the authors continue working on this field, see modelInt/td>
Cloudfier	Free trial	Rapid Application Development from textual UML models (see TextUML) including static and dynamic (action-based) specifications



Abstract Solutions xUML	Commercial	Abstract Solutions is a new incarnation of the company previously known as Kennedy-Carter, also deriving from the original proposals for Executable UML.
QM	freeware	Lightweight UML modeling tool for designing and implementing real-time embedded applications based on the QP state machine frameworks. Generates compact and efficient C or C++ code suitable for single-chip microcontrollers. Extended notation for internal state transitions
Sinelabore	Commercial	Command line tool for code-generation from UML state machines, especially targeting resource limited embedded real-time and high-availability systems. It has simulation, tracing and test-case generation capabilities.
EM/OS Enterprise Model Operation Services	Commercial, partly Open Source	Fully operational business applications, all tiers (Java) created from single, annotated, UML based model; Standard compliance desired, currently difficult due to abstraction level differences



Matrix	Free trial	Abstract modeling language and Model Compiler featuring full automatic code generation and interactive Simulator. See also this intro
txtUML	Open Source	txtUML stands for textual, executable and translatable UML. txtUML models can be run, debugged and tested using the standard Java runtime environments,It includes a compiler for C++
Umple	Open Source	Umple is a modeling tool and programming language family to enable what they call Model-Oriented Programming. Read more



Older tools (now discontinued)

Name	License	Comments
LieberLieber AM|USE 2.0	Commercial	Offers both state machine and activity model execution. Integrated in Sparx Enterprise Architect
e-Alf	Apache License 2.0	Eclipse Implementation of Action Language for Foundational UML using: Eclipse UML2 xtext Acceleo ATL. No activity in the last months
TOPCASED Model Simulation	EPL	This tool has been merged into Papyrus (see above).
Lohr	GNU Lesser GPL	A high level programming and modeling language for creating executable models of software systems
UML Almighty	Open source	UML tool capable of execute behavior and generate a Web prototype to execute that behavior.
Pathfinder Solutions PathMATE	Commercial	PathMATE transforms (executable) platform independent models to “efficient, high-performance” code.

From: https://modeling-languages.com/text-uml-tools-complete-list/

Textual UML tools

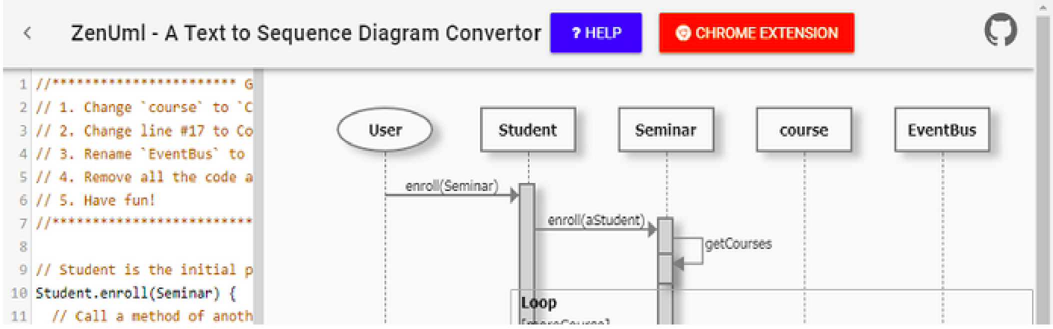
Sometimes old plain text triumphs over nice cool graphics. If you think graphical UML editors are too slow or cumbersome, there are plenty of textual UML tools available (and some of them rather successful).

All textual UML tools offer some kind of (mostly) simple language to describe your UML class, sequence, activity ,… diagrams. You can use this textual UML description to store and compare your models but you can still visualize teh resulting diagram since all of them are able to automatically display the corresponding graphical UML diagram from its textual description.

Wanna know more? see our complete list of textual modeling tools.

Text to UML tools – Fastest way to create your models

by Jordi Cabot | Dec 28, 2018 | tools, UML and OCL | 6 comments



```
//*********************** G
// 1. Change `course` to `C
// 2. Change line #17 to Co
// 3. Rename `EventBus` to
// 4. Remove all the code a
// 5. Have fun!
//*************************

// Student is the initial p
Student.enroll(Seminar) {
  // Call a method of anoth
```

A textual UML tool supports the use of textual notations/languages to describe UML models and automatically renders the corresponding graphical UML diagram from that textual description (a few tools also target other kinds modeling languages, like ER or BPMN, and we mention them here as well, but they are the exception).

The textual UML tools market is one of the fastest growing segment in the UML tools market (based on my own perception of visitors' interest). Together with online modeling tools, they are the go-to option for all people looking for some kind of lightweight solution to draw some models. In fact, since most textual UML tools have an online editor, they are a jackpot for occasional modelers.

But why are text-to-UML tools so popular? The short answer is that **textual modeling tools have a very low barrier to entry**. The fact that UML models are stored as text simplifies their integration with a variety of tools (like version control systems) that programmers already use in their everyday work so there's no need to learn/buy/install additional tools. And **programmers typically feel more comfortable with textual languages than with graphical ones**. Both aspects represent a huge boost for the adoption of these tools.

Still, as you'll see in the list below, features, expressiveness, and robustness of such tools are rather limited in most cases. That's why I was saying that these tools are more of an option for quick and dirty model sketches (for documentation or blueprints for early design discussions) more than a serious and deep modeling activity. I wonder if this typical usage scenario is what discourages company to (barely) offer any kind of commercial solution for this market.

> Complete list of online tools to render UML models from a few lines of text
>
> CLICK TO TWEET

Let's see our complete list of text to UML tools. I tried to include **all** tools I'm aware of. If you think yours is missing please leave a comment and I'll add it. Note that I'm listing here end-user modeling tools. If you're looking for (JavaScript) libraries that help you to create your own modeling editor, go here.

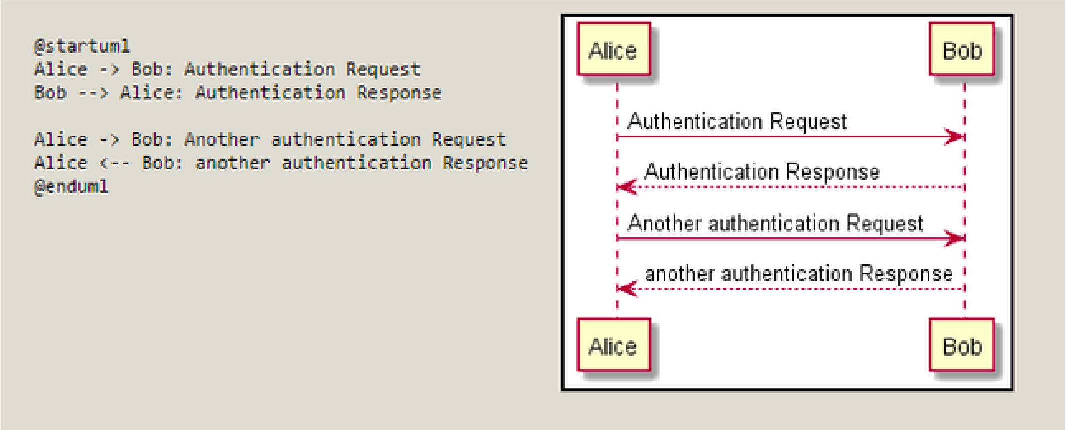
Most promising UML textual modeling tools

In no particular order, these are the tools you should check first when looking for a quick and easy way to draw some UML diagrams. As long as you are interested in drawing class diagrams, sequence diagrams or use case diagrams you'll find several options. A couple also support state machines. Coverage of other kinds of UML diagrams is rather poor.

PlantUML

PlantUML is the most well-known UML tool in this category with millions of UML rendered. We have covered it in-depth in this interview with his creator but, in short, it supports all important UML diagrams (class, use case, activity, sequence, component, deployment and object diagrams but, to me, the strong point of this tool is the variety of scenarios in which can be used. There's a huge ecosystem of tools around PlantUML to render textual UML diagrams anywhere you want.

PlantText UML Editor embeds PlantUML in a live online editor.



```
@startuml
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response

Alice -> Bob: Another authentication Request
Alice <-- Bob: another authentication Response
@enduml
```

Alice
Bob
Authentication Request
Authentication Response
Another authentication Request
another authentication Response
Alice
Bob

Simple PlantUML Sequence diagram

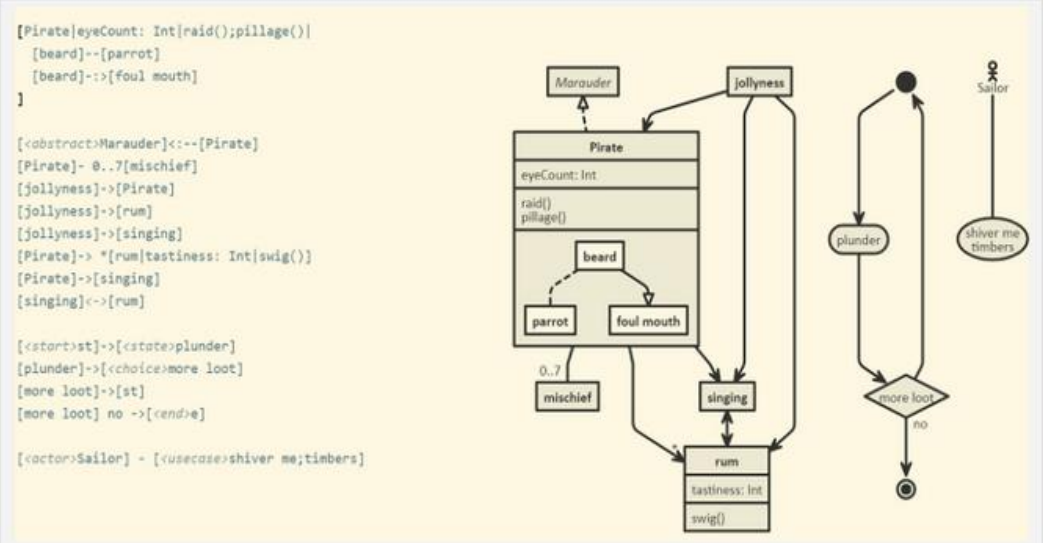
yUML

yUML is an online service for creating class and use case diagrams, with activity diagrams and state machines anounced to come soon. It's makes it really easy for you to: Embed UML diagrams in blogs, emails and wikis, post UML diagrams in forums and blog comments, use directly within your web basec bug tracking tool or copy and paste UML diagrams into MS Word documents and Powerpoint presentations.

The service can be called from your blog or web page (with the textual description as part of the URL) tc automatically display the image when accessing it. As paid options, you can use your own namespace fc the images or even install it on your own host. Several integrations with third-party tools are also available.

Nomnoml

The nomnoml web application is a simple editor with a live preview. It is purely client side and changes are saved to the browser's *localStorage*, so your diagram should be here the next time (but no guarantees). You can also the nomnoml standalone javascript library to render diagrams on your own web page. Find the source code on GitHub.



```
[Pirate|eyeCount: Int|raid();pillage()|
  [beard]--[parrot]
  [beard]-:>[foul mouth]
]

[<abstract>Marauder]<:--[Pirate]
[Pirate]- 0..7[mischief]
[jollyness]->[Pirate]
[jollyness]->[rum]
[jollyness]->[singing]
[Pirate]-> *[rum|tastiness: Int|swig()]
[Pirate]->[singing]
[singing]<->[rum]

[<start>st]->[<state>plunder]
[plunder]->[<choice>more loot]
[more loot]->[st]
[more loot] no ->[<end>e]

[<actor>Sailor] - [<usecase>shiver me;timbers]
```

nomnoml live editor

TextUML

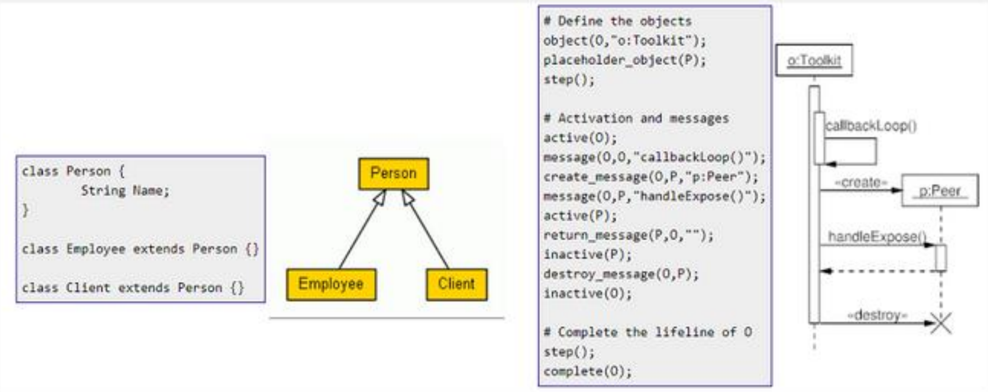
TextUML Toolkit is an open-source IDE for UML to create models at the same speed you write code, therefore, offering increased modeling productivity. TextUML is compatible with all tools that support Eclipse UML2 models. TextUML offers all features you like in your favorite IDE: instant validatic syntax highlighting, outline view, textual comparison and live graphical visualization of your model as c diagrams

The TextUML Toolkit can be used both as a set of plug-ins for the Eclipse IDE, and as a part of a multi-tenant server-side application – as seen in Cloudfier.

While the last version dates from 2015, it remains a go-to tool for textual modeling within the Eclipse community.

UML Graph

UML Graph automatically renders class and sequence diagrams. For the class diagrams, it uses a Java-based syntax complemented with javadoc tags. Running the UmlGraph doclet on the specification will generate a Graphviz diagram specification. For sequence diagrams, UMLGraph uses a different approach (and this is one aspect I don't like about the tool, you are basically working with two different tools here). Pic macros are used to define objects and method invocations. Then, the pic2plot program processes the macros to generate PNGs and other graphics formats. LightUML integrates UMLGraph in Eclipse. UMLGraph in GitHub.



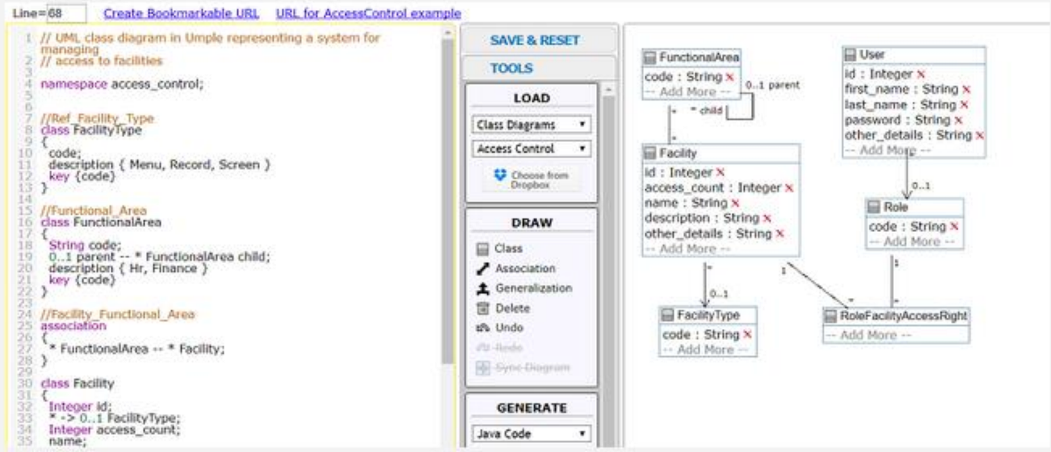
```
class Person {
        String Name;
}

class Employee extends Person {}

class Client extends Person {}
```

```
# Define the objects
object(O,"o:Toolkit");
placeholder_object(P);
step();

# Activation and messages
active(O);
message(O,O,"callbackLoop()");
create_message(O,P,"p:Peer");
message(O,P,"handleExpose()");
active(P);
return_message(P,O,"");
inactive(P);
destroy_message(O,P);
inactive(O);

# Complete the lifeline of O
step();
complete(O);
```

UMLGraph example

Umple

Umple can also be used as a textual modeling tool for UML even it is aimed at a slightly different purpose Umple merges the concepts of programming and modeling by adding modeling abstractions directly into programming languages. Currently, Umple supports Java, PHP and Ruby as base languages. It adds UML attributes, associations and state machines to these languages. Read our post on Umple for more details on the history and background of Umple.

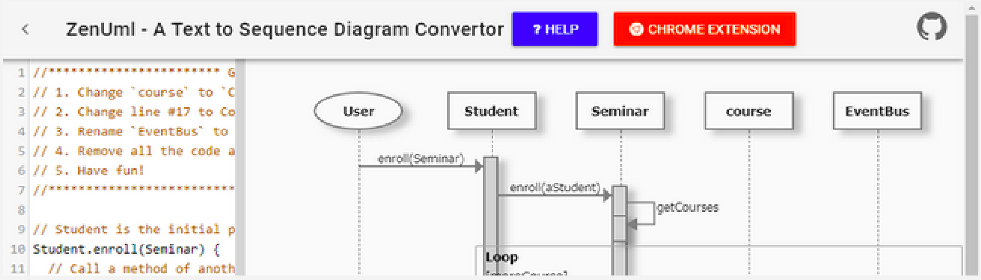


```
// UML class diagram in Umple representing a system for managing
// access to facilities

namespace access_control;


//Ref_Facility_Type
class FacilityType
{
  code;
  description { Menu, Record, Screen }
  key {code}
}

//Functional_Area
class FunctionalArea
{
  String code;
  0..1 parent -- * FunctionalArea child;
  description { Hr, Finance }
  key {code}
}

//Facility_Functional_Area
association
{
  * FunctionalArea -- * Facility;
}

class Facility
{
  Integer id;
  * -> 0..1 FacilityType;
  Integer access_count;
  name;
```

UmpleOnline model editor

ZenUML

ZenUML is one of the latest tools to enter the market. Read why the author believed that ZenUML was needed when there were already so many other textual tools for UML sequence diagrams. In short, creating sequence diagrams with ZenUML is really fast even for complex diagrams.

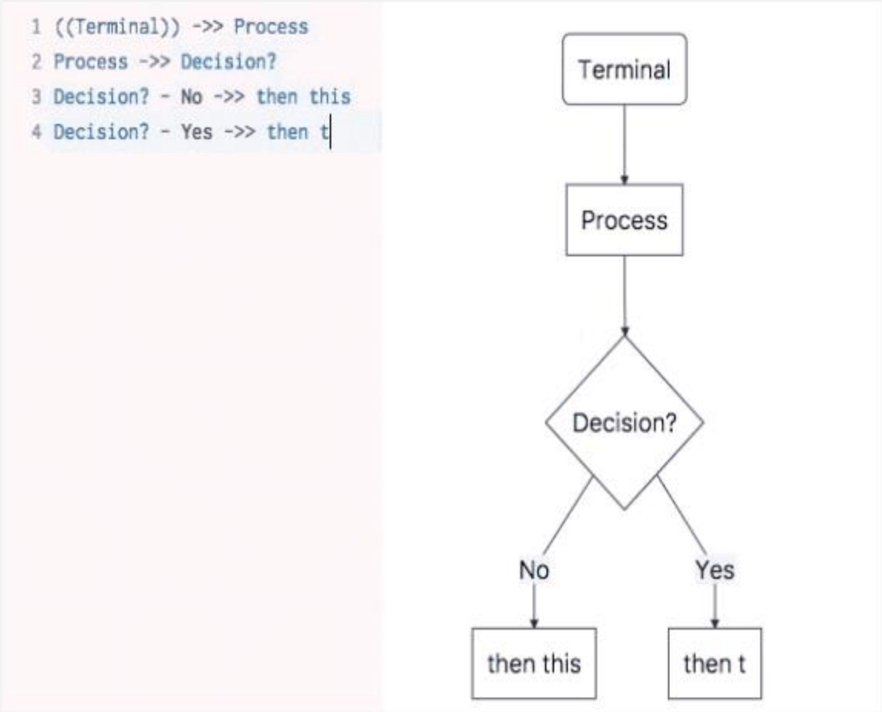


```
//*********************** G
// 1. Change `course` to `C
// 2. Change line #17 to Co
// 3. Rename `EventBus` to
// 4. Remove all the code a
// 5. Have fun!
//*************************

// Student is the initial p
Student.enroll(Seminar) {
  // Call a method of anoth
```

ZenUML editor for sequence diagrams

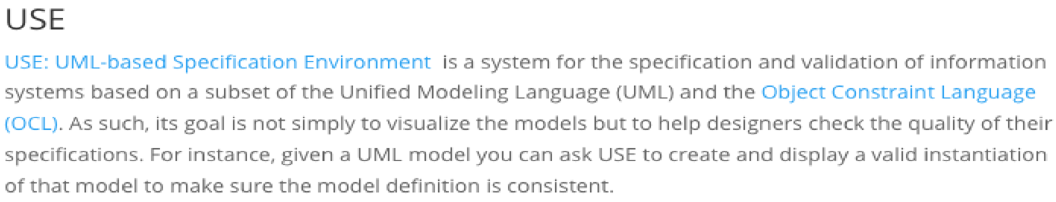
Chart Mage

Chart Mage enables the creation of flowcharts and sequence diagrams. As we describe here in more detail, the main feature of Chart Mage is its autocomplete functionality that makes a reasonable guess what you're going to write next based on the typical UML syntax and the partial model you've created s far.



```
((Terminal)) ->> Process
Process ->> Decision?
Decision? - No ->> then this
Decision? - Yes ->> then t
```

Chart Mage in action: from text to flowcharts and UML sequence diagrams

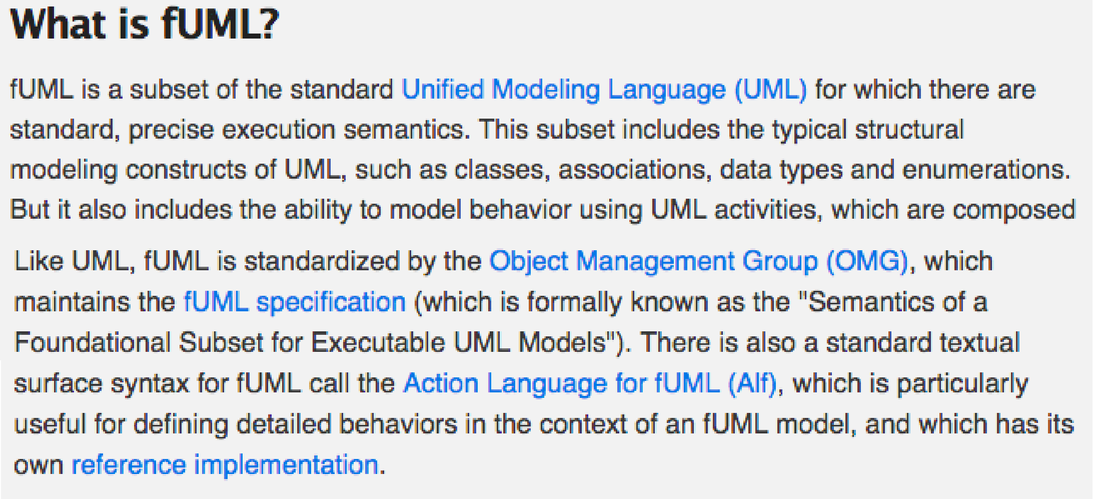


USE

USE: UML-based Specification Environment is a system for the specification and validation of information systems based on a subset of the Unified Modeling Language (UML) and the Object Constraint Language (OCL). As such, its goal is not simply to visualize the models but to help designers check the quality of their specifications. For instance, given a UML model you can ask USE to create and display a valid instantiation of that model to make sure the model definition is consistent.

8.2 fUML

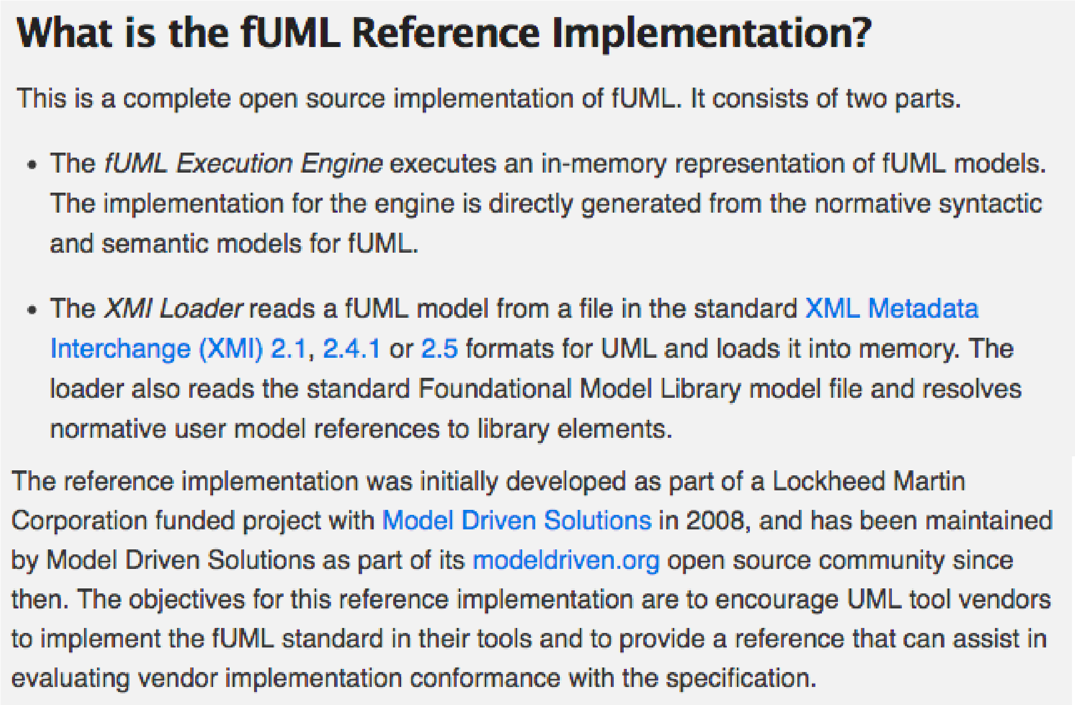
Extracts from: https://www.omg.org/spec/FUML/1.4 (December 2018).



What is fUML?

fUML is a subset of the standard Unified Modeling Language (UML) for which there are standard, precise execution semantics. This subset includes the typical structural modeling constructs of UML, such as classes, associations, data types and enumerations. But it also includes the ability to model behavior using UML activities, which are composed

Like UML, fUML is standardized by the Object Management Group (OMG), which maintains the fUML specification (which is formally known as the "Semantics of a Foundational Subset for Executable UML Models"). There is also a standard textual surface syntax for fUML call the Action Language for fUML (Alf), which is particularly useful for defining detailed behaviors in the context of an fUML model, and which has its own reference implementation.

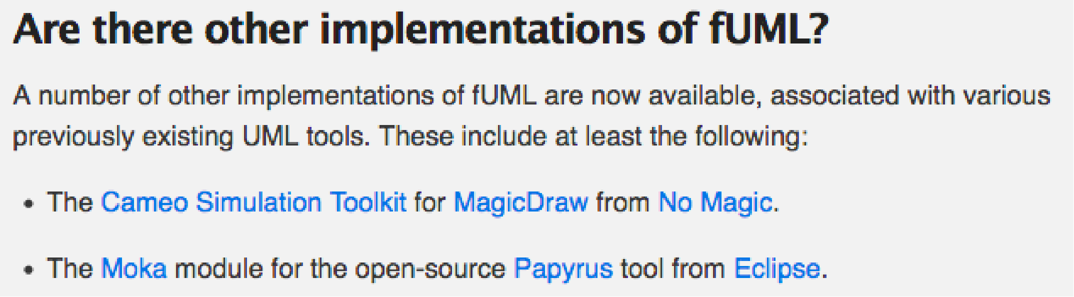


What is the fUML Reference Implementation?

This is a complete open source implementation of fUML. It consists of two parts.

- The *fUML Execution Engine* executes an in-memory representation of fUML models. The implementation for the engine is directly generated from the normative syntactic and semantic models for fUML.
- The *XMI Loader* reads a fUML model from a file in the standard XML Metadata Interchange (XMI) 2.1, 2.4.1 or 2.5 formats for UML and loads it into memory. The loader also reads the standard Foundational Model Library model file and resolves normative user model references to library elements.

The reference implementation was initially developed as part of a Lockheed Martin Corporation funded project with Model Driven Solutions in 2008, and has been maintained by Model Driven Solutions as part of its modeldriven.org open source community since then. The objectives for this reference implementation are to encourage UML tool vendors to implement the fUML standard in their tools and to provide a reference that can assist in evaluating vendor implementation conformance with the specification.

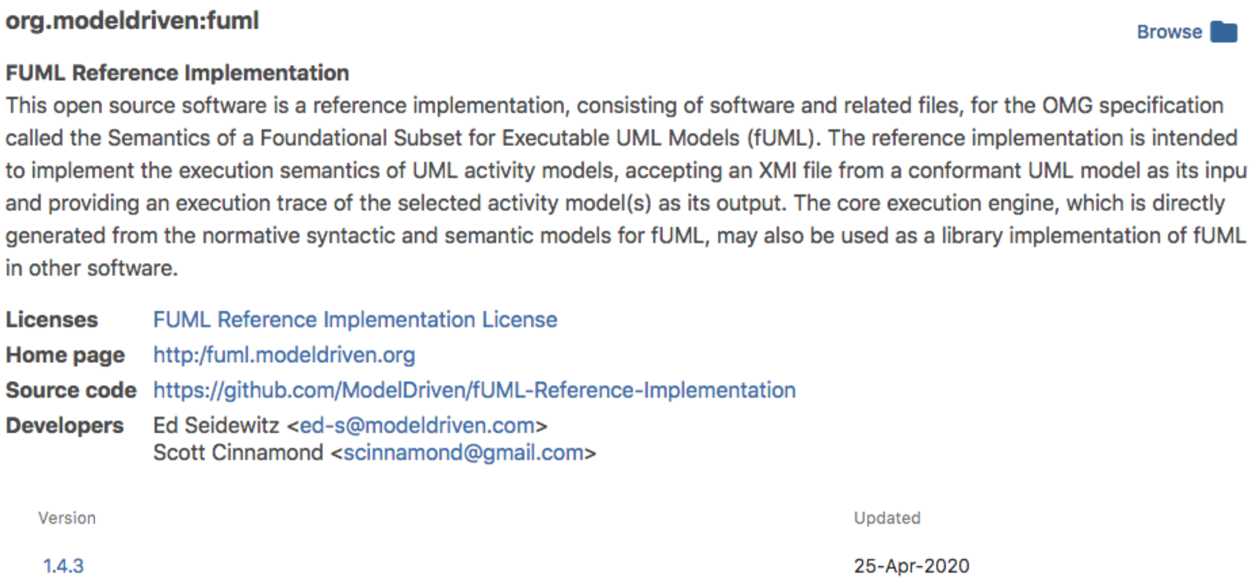


Are there other implementations of fUML?

A number of other implementations of fUML are now available, associated with various previously existing UML tools. These include at least the following:

- The Cameo Simulation Toolkit for MagicDraw from No Magic.
- The Moka module for the open-source Papyrus tool from Eclipse.

Extracts from:
https://github.com/ModelDriven/fUML-Reference-Implementation/blob/master/README.md
(April 2020)



org.modeldriven:fuml

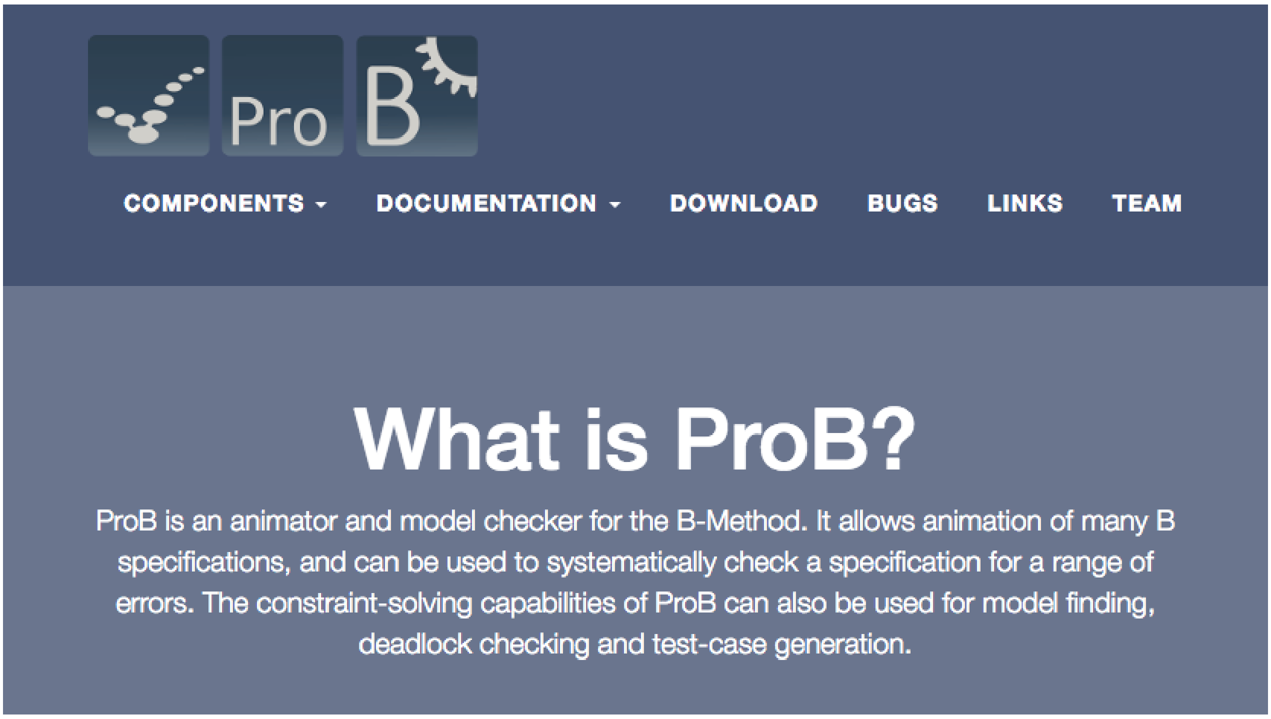
FUML Reference Implementation
This open source software is a reference implementation, consisting of software and related files, for the OMG specification called the Semantics of a Foundational Subset for Executable UML Models (fUML). The reference implementation is intended to implement the execution semantics of UML activity models, accepting an XMI file from a conformant UML model as its inpu and providing an execution trace of the selected activity model(s) as its output. The core execution engine, which is directly generated from the normative syntactic and semantic models for fUML, may also be used as a library implementation of fUML in other software.

Licenses	FUML Reference Implementation License
Home page	http:/fuml.modeldriven.org
Source code	https://github.com/ModelDriven/fUML-Reference-Implementation
Developers	Ed Seidewitz <ed-s@modeldriven.com> Scott Cinnamond <scinnamond@gmail.com>

Version	Updated
1.4.3	25-Apr-2020

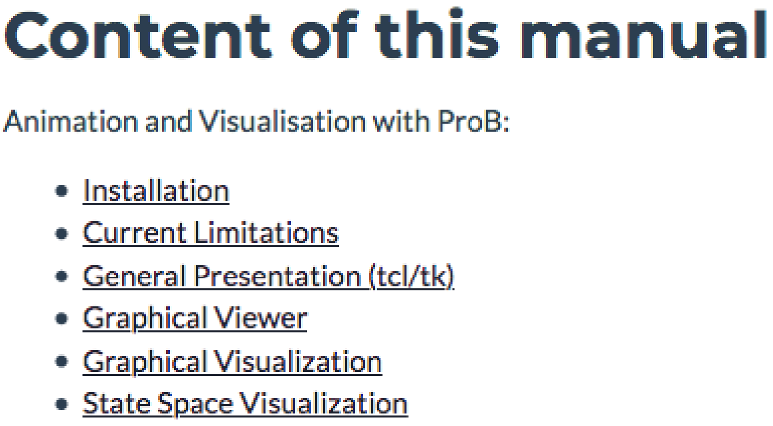
8.3 ProB

from: https://www3.hhu.de/stups/prob/
https://www3.hhu.de/stups/handbook/prob2/prob_handbook.pdf



What is ProB?

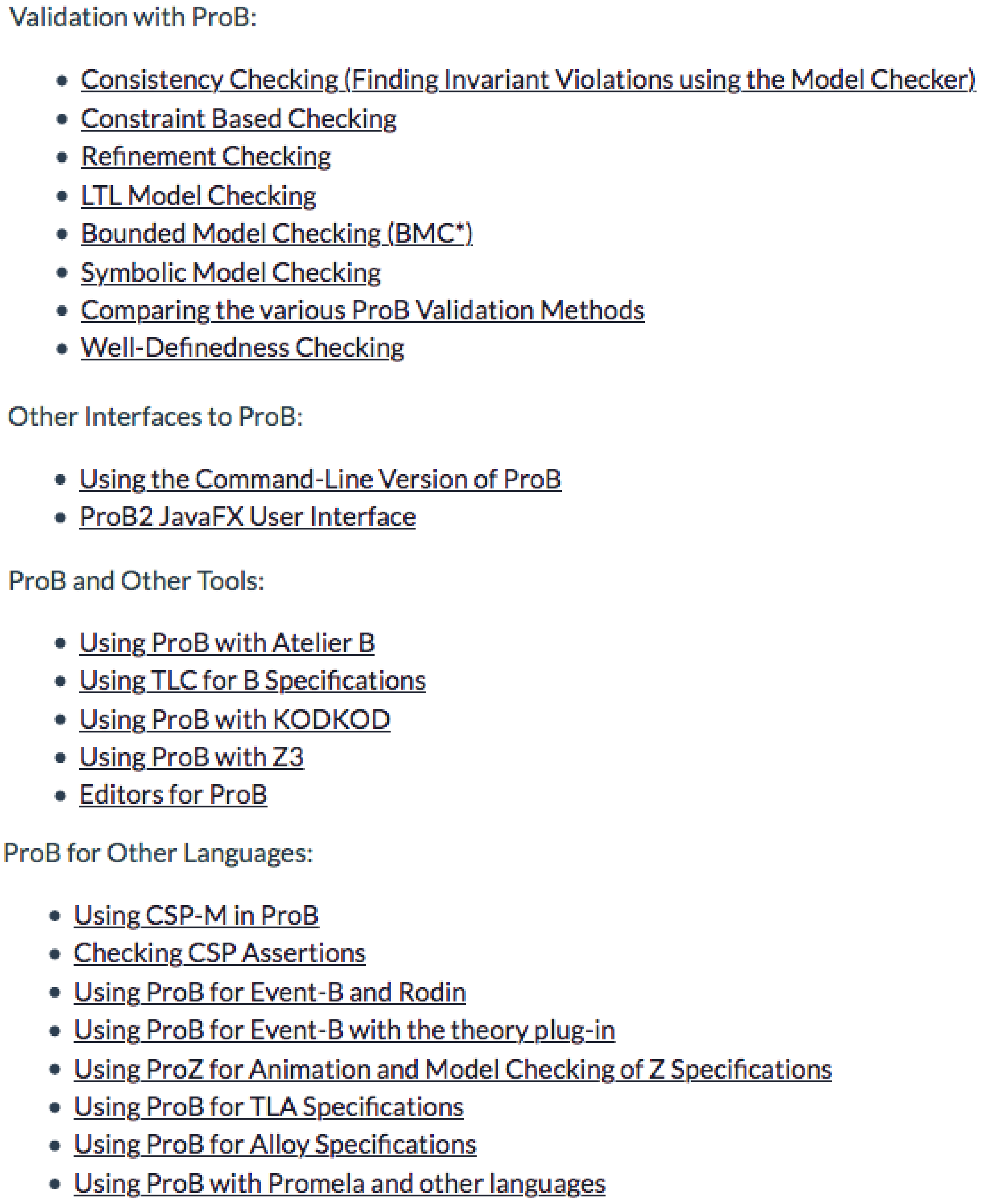
ProB is an animator and model checker for the B-Method. It allows animation of many B specifications, and can be used to systematically check a specification for a range of errors. The constraint-solving capabilities of ProB can also be used for model finding, deadlock checking and test-case generation.



Content of this manual

Animation and Visualisation with ProB:

- Installation
- Current Limitations
- General Presentation (tcl/tk)
- Graphical Viewer
- Graphical Visualization
- State Space Visualization



Validation with ProB:

- Consistency Checking (Finding Invariant Violations using the Model Checker)
- Constraint Based Checking
- Refinement Checking
- LTL Model Checking
- Bounded Model Checking (BMC*)
- Symbolic Model Checking
- Comparing the various ProB Validation Methods
- Well-Definedness Checking

Other Interfaces to ProB:

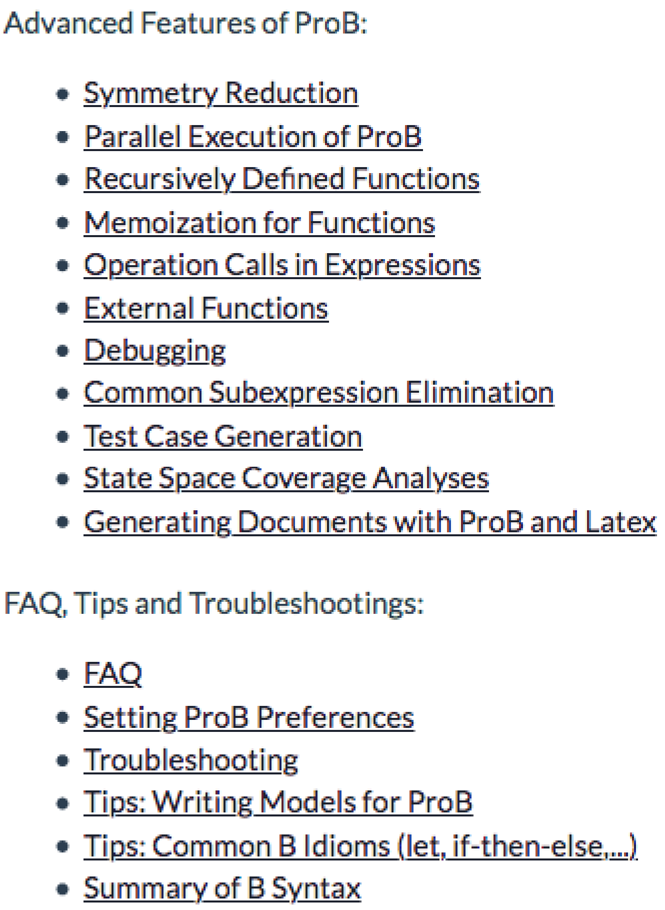
- Using the Command-Line Version of ProB
- ProB2 JavaFX User Interface

ProB and Other Tools:

- Using ProB with Atelier B
- Using TLC for B Specifications
- Using ProB with KODKOD
- Using ProB with Z3
- Editors for ProB

ProB for Other Languages:

- Using CSP-M in ProB
- Checking CSP Assertions
- Using ProB for Event-B and Rodin
- Using ProB for Event-B with the theory plug-in
- Using ProZ for Animation and Model Checking of Z Specifications
- Using ProB for TLA Specifications
- Using ProB for Alloy Specifications
- Using ProB with Promela and other languages



Advanced Features of ProB:

- Symmetry Reduction
- Parallel Execution of ProB
- Recursively Defined Functions
- Memoization for Functions
- Operation Calls in Expressions
- External Functions
- Debugging
- Common Subexpression Elimination
- Test Case Generation
- State Space Coverage Analyses
- Generating Documents with ProB and Latex

FAQ, Tips and Troubleshootings:

- FAQ
- Setting ProB Preferences
- Troubleshooting
- Tips: Writing Models for ProB
- Tips: Common B Idioms (let, if-then-else,...)
- Summary of B Syntax