An Extended Survey of Open Source Model-Based Engineering Tools

Report prepared for Ericsson by:

Kenn Hussey, Bran Selic, and Toby McClean

Revision E (May 11, 2010)

Zeligsoft 115, rue Principale, Suite 300 Gatineau, QC J9H 3M2 **T** (819) 684-9639 **F** (819) 684-5249 http://www.zeligsoft.com

Introduction

This report presents the results of a study that was undertaken, by Zeligsoft on behalf of Ericsson, to investigate the current state of the art and trends in the domain of open source modeling tools and capabilities. The study had the following objectives:

- To describe the different open source projects, tools, and capabilities that are currently available, with special focus on the Eclipse environment;
- To identify the state of the different projects, tools, and capabilities
- To identify which organizations are involved in developing open source capabilities in this domain; and
- To identify which organizations are using open source capabilities in this domain.

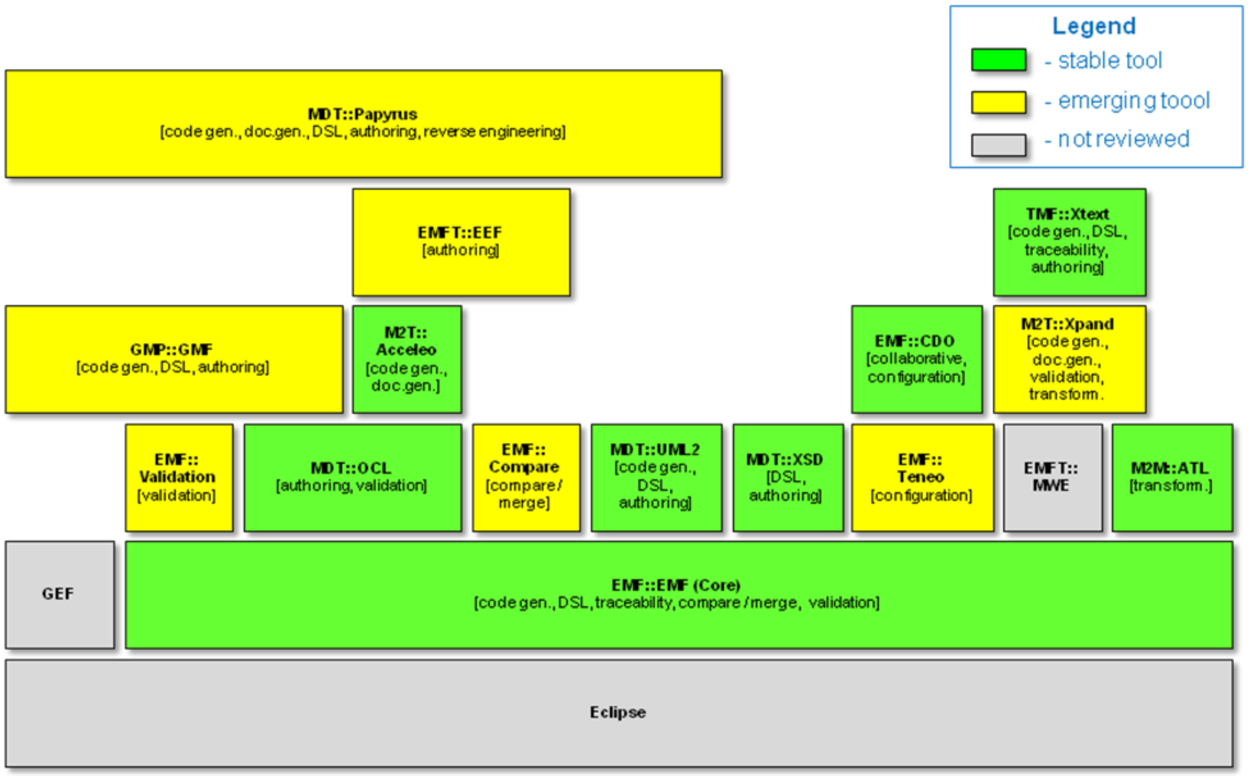
Executive Summary

During the analysis phase, it became apparent that there is significant fragmentation and duplication of effort in the domain of open source modeling tools, even within the narrow context of the Eclipse ecosystem where some of the technology is fairly mature. Consequently, it seems apparent that a major consolidation is needed to ensure a critical mass of contributors required for long-term survival and evolution of the various projects; as things currently stand, there is some concern as to the availability of long-term support for key capabilities.

There is also a noticeable shift from vendor-driven to end-user-driven development in open source; this shift will require strong end-user participation to be viable in the long term. Evidence that key tool vendors are rethinking their commitments to open-source can be gleaned from the recent reduced level of support by IBM for two key modeling subprojects (EMF and the UML 2 Eclipse projects), as well as Borland's decision to cease contributing to the UML2 Tools initiative. At the same time, we are seeing a number of large end-user enterprises (SAP, UBS, Airbus, etc.) combining forces and stepping up their participation and commitments to open-source modeling tools. Representative initiatives that seem most promising include OPEES, the Eclipse Modeling Platform Working Group, and recently submitted Eclipse project proposals such as Graphiti, Sphinx, and the EMF Client Platform (ECP). It is worth noting that the failure of such user-led initiatives is likely have a major negative impact on the broader adoption of model-based methods by the general development community, which will, consequently, have negative repercussions for those enterprises that are committed to and have come to depend on this approach to software development.

Architecture

In principle, it should soon be possible to form a modeling tool platform by selecting stable modeling projects from Eclipse, leveraging work from the recently proposed Sphinx project and/or the emerging Modeling Platform Working Group, and building remaining high-level end user functionality on top, perhaps using Papyrus or UML2 Tools as a starting point. A dependency architecture based on the most suitable of the existing modeling projects (based on the evaluations described in more detail below) might take on the form depicted in the following dependency diagram (projects higher in the stack depend on projects lower in the stack). Green and yellow shading indicates the level of stability/quality (green indicates greater stability than yellow), as per the chosen evaluation criteria; while items shaded in grey were not evaluated as part of this study.



Roadmap

We feel it is unrealistic to expect a suitable open source modeling tool platform to be available any sooner than mid-2011, and then only with the necessary amount of investment and leadership. In this case, "leadership" implies not only strong technical guidance but also responsibility from both product and project perspectives. Consequently, the aforementioned initiatives such as OPEES and the Eclipse Modeling Platform Working Group are crucial to the success of such a vision.

Of course, a more accurate and detailed roadmap can only be produced after doing a proper gap analysis, resource estimation, and project planning.

Method

Phases

The study was divided into the following sequentially ordered phases:

Phase 1
This phase was focused on determining the scope of study. It was decided that the study should focus on Eclipse-based frameworks and tools (and, in particular, projects at Eclipse) with cursory treatment of other open source technologies.

Phase 2
This phase involved identification of key/desirable functionality that could be used to form a high-level architecture/vision of Eclipse with respect to modeling frameworks and tools. Basic capabilities were identified, as were the important evaluation criteria for assessing the frameworks and tools that were determined to be in scope for the study. A subset of the many available open source projects were selected, based on their support for the chosen capabilities, and sources were identified for gathering the necessary information to evaluate the projects based on the criteria.

Phase 3
In this phase, information was gathered for each of the relevant projects in order to assess their suitability for inclusion in a modeling tool platform to support the key capabilities. Members of the projects' communities (developers, vendors/consumers, users) were identified, as were their relation to open specifications and/or standards. Where overlap existed between projects, key differentiators were identified.

Phase 4
This phase involved the formation of recommendations on whether/how to consume the evaluated projects based on their assessments. These recommendations are presented herein.

Capabilities

Model-based design and development involves a myriad of different specialized frameworks and tools, and, in the context of this study, it is not realistic to cover the full set. Instead, we focus on capabilities that we feel are of primary interest. We use the term “capability” instead of tool or framework, since it is usually the case that tools and/or frameworks combine multiple capabilities. We identify two categories of such capabilities.

Primary Capabilities

These are essential to the design and development process, that is, they are capabilities without which it is not reasonable to consider doing model-based development. These include the following:

Authoring
This is the ability to create primary development artifacts, such as models, code, transforms, documents, etc., depending on the tool and the domain. For example, for a UML tool, this is the ability to create and update UML models, whereas for a model-transformation design tool, it is the ability to create and update model transforms.

Code Generation
This includes capabilities for automatic code generation from implementation models. An important requirement related to these is that the code generation can be customized.

It should be possible to do advanced and open code generation from the modeling tool, including the ability to generate towards arbitrary target languages with arbitrary optimizations. The technology used to provide this code generation support should be intuitive to use and not require advanced programming skills. Rather, the technology should work with the underlying metamodel in a natural and open way.

The code generation technology should scale to very large models and provide a good environment for transformation development, e.g., the ability to debug the transforms.

Collaborative Development
The framework or tool should support team and collaboration concepts that scale to very large models with many concurrent users across different sites and time-zones.

Compare / Merge
This capability covers support for doing seamless logical three-way differencing and merging of models without being aware of underlying metamodel or other extraneous information related to layout or persistence. It should be possible to configure and filter the information that is presented while comparing and merging models.

Configuration Management
This comprises configuration management capabilities for maintaining multiple controlled versions of models and for generating executables which incorporate, among other elements, code generated from those models. This includes ancillary capabilities such as model compare and merge and model persistence.

It should be possible to manage configurations of model artifacts from a very coarse to a very fine level of granularity, and this functionality should be independent of the underlying persistence technology.

Debug & Trace
This covers the ability to debug, visualize, and verify target execution from within a model, i.e., model-level features for integrating and troubleshooting a real target.

The modeling environment and any code or artifacts generated or produced should integrate seamlessly with underlying trace and debug features provided by Eclipse (e.g., watches, various kinds of breakpoints, execution alteration, etc.).

Document Generation
This is the ability to automatically derive documentation about an artifact created and/or maintained by a modeling tool. Usually, it refers to the ability to generate design documents and reports according to formats specified by the end user organization.

DSL Support
This includes support for creating both graphical and textual domain specific languages (DSLs). The mechanism(s) provided for doing this should be intuitive to use and integrate coherently with the rest of Eclipse

When creating a DSL, it should be possible to expose in the end user experience only those concepts that are relevant to the DSL. Eclipse already has diverse support for this today, but more needs to be done to package and simplify these mechanisms.

Review
This involves support for doing model reviews and adding comments to a model, without polluting the underlying application model. This should include the ability to track changes and updates that result from a review comment.

Simulation
In order to support system engineers, there is a need for the ability to do partial model simulation, i.e., to define a partially complete model and simulate its behavior. Behavior would be derived from artifacts such as state machines, activity diagrams, and sequence diagrams, and would be augmented by an action language.

One use case here is to enable a system engineer to model and simulate behavior without the need for advanced programming skills. This way, each phase of design abstraction encompasses simulation/execution, testing, and debugging at an appropriate level of sophistication.

Testing
Support for building test specification models and then generating deployable test cases. This corresponds closely with capabilities for DSL support and code generation, but with an emphasis on algorithms for different kinds of test coverage.

Traceability
Tracking model artifact evolution from a specification in a system model to design and implementation in a design model. The guiding philosophy here is that, given a design artifact, it should be possible to trace back to the originating artifact in a design model or a requirement. Of course, the reverse should be possible if a design model is already in place.

This concept could be extended to any number of abstraction levels and could perhaps be broadened to additionally support inter-model (and inter-tool) linking.

Transformation
This refers to model-to-model transformations of various kinds, typically from one modeling language, based on one metamodel, to another, based on a different metamodel. This is used in a variety of ways such as translating a UML model into a schedulability model to compute expected timing characteristics, to generate different views of a given model for different viewpoints, or to generate more abstract models from concrete ones. Note that this capability does NOT include code generation, since that is a separate category (see above).

Validation
This is the ability to perform various kinds of automated checks for consistency and correctness, usually based on some kind of formalism. This includes conformance to the formal rules of the modeling language, as well as possible user-defined checks (e.g., to detect presence of anti-patterns).

UML Action Language Support
This capability is essentially an implementation of the OMG’s UML action language (currently under development), to at least the first (Basic) level of conformance.

Adopting a philosophy that “the model is the code” but then polluting models with C or C++ code tends to result in models with an unsuitable mix of abstraction levels. Far higher productivity could be achieved by using a simple action language to specify behavior while relying on parts of UML or an alternative domain specific language to define structure and behavior. Transformations would then used to produce deployment code for a chosen target language and platform.

Secondary Capabilities

These are capabilities that are not essential but which still have the potential to significantly improve developer productivity and product quality. These include the following:

Assets

This is the ability to classify and index development-related reusable assets of various kinds (e.g., models, documentation, design patterns, code fragments) in a manner that makes them easy to discover and retrieve. A prototypical example of this is the Eclipse-based Reusable Assets Specification (RAS) adopted by the OMG.

Reverse Engineering

This is the ability to extract useful models from program language source code. It is a feature that is essential in round-trip engineering type development (complemented, of course, by code generation), but is also used in re-engineering of legacy systems and documentation.

Criteria

Each project that provided one or more of the capabilities described above was evaluated based on a set of criteria, described below. Beyond these criteria, we include a general description of the project, the licenses under which is it made available, and the platforms and/or languages it supports, and provide some guidance as to whether/how to consume them (especially in cases of overlap with other technologies).

Business Criteria

One angle from which to evaluate the open source landscape is the business side of things, and many of the readily available sources of information on a project are geared towards this kind of evaluation. These are generally easy to measure, since they lend themselves to more of a quantitative analysis. These criteria are based on who the members of the various kinds of communities are (developers, vendors/consumers, users) and what they are doing, and could perhaps be considered a measure of stability. They include the following:

Predictability

This is a measure of the maturity and stability of the framework or tool, including whether the project is participating in, or has participated in, coordinated simultaneous releases of Eclipse (and if so, which ones) and the release timeline for the project, if available.

Committer Community

A thriving, diverse and active community of developers is the key component of an open source project. Ideally, this community should be an open, transparent, inclusive, and diverse community of committers and (non-committer) contributors. Attracting new contributors and committers to an open source project is time consuming and requires active recruiting, not just passive "openness". The project leadership must make reasonable efforts to encourage and nurture promising new contributors.

Key metrics here include the following:

Diversity. Number of participants (companies) over time and who they are.

Responsiveness. Number (or percentage) of bugs fixed over time. Number (or percentage) of newsgroup/forum posts answered over time.

Activity. Number of commits over time and who key contributors are.

Openness. Mailing list activity over time.

User Community
An active and engaged user community is proof-positive that a project's exemplary frameworks and/or tools are useful and needed. Furthermore, a large user community is one of the key factors in creating a viable ecosystem around an open source project, thus encouraging additional open source and commercial organizations to participate. Like all good things, a user community takes time and effort to bring to fruition, but, once established, is typically self-sustaining.

Key metrics here include:

Downloads. Number of downloads over time (and, if possible, who they are - may be by geography only).

Involvement. Number of bugs opened over time (and, if possible, by whom). Number of newsgroup/forum posts over time (and, if possible, by whom).

For the purposes of this study, we did not always focus on generally accessible user information (e.g., such as that obtainable by searching the Web with Google), but, for Eclipse projects at least, attempted to gather “inside” information based on download statistics and input provided by project leads.

Adopter Community
An active and engaged adopter/plug-in developer community is the only way to prove that an open source project is providing extensible frameworks and exemplary tools accessible via documented APIs. Reuse of the frameworks within the companies that are contributing to the project is necessary, but not sufficient, to demonstrate an adopter community. Again, creating, encouraging, and nurturing an adopter community outside of the project's developers takes time, energy, and creativity by the project leadership, but is essential to the Project's long-term open source success.

Key metrics here include the following:

Internal. Number (or percentage) of committers/contributors that have reused the framework/tool.

External. Number of external adopters (if/where possible to determine).

Learning
Learning resources that are available for the project, including documentation and a list of providers of education/training.

Support
Support resources that are available for the project, including support forums and a list of providers of consulting/services.

Other Criteria

The other criteria used in this study were related to technology, i.e., the actual software/architecture and are, perhaps, harder to measure in the absence of empirical data. These criteria could perhaps be considered a measure of quality. They include the following:

Customizability
That is, the ability to use the project and its existing facilities in ways that are best suited to the task at hand. This principle covers several distinct areas of concern:

Tools. The project should enable users to choose which tools they want to use and the way in which they want to use them.

Processes. Although the project may be targeting a particular kind of process, it should be possible to customize such processes to individual needs or even to use completely different processes.

Languages. In general, the project must support flexibility in selecting specification and implementation languages best suited to the problem on hand.

Methods. The choice of analysis, design, testing and other methods should not be pre-ordained by the framework or tool, but left up to individual projects and users.

Extensibility
This is the ability to extend the framework/tool with new capabilities including those that may not have been anticipated during the original conception. Again, this principle is manifested in the key areas noted above:

Tools – the addition of new tools

Processes – support for new process models

Languages – support for new modeling languages as well as the ability to define new custom languages

Methods – the addition of new analyses methods, testing methods, etc.

Scalability
This is the ability of the toolset to cover a broad range of differently sized problems. This includes not only the complexity and size of the system being developed but also the ability to scale up to large development teams and environments. The toolset should be capable of supporting simple single-person projects to ones requiring globally distributed development teams incorporating hundreds of designers.

Usability
As an enabling technology for building complex tools, Eclipse is considered top-class. However, when using an Eclipse-based modeling tool, different modes of operation are generally necessary (but often lacking). One of the biggest risks in large scale adoption of Eclipse technologies for specialized domains is the amount of "noise" thrown at the user.

A good project must minimize the cognitive load on users stemming from accidental complexity of the framework/tool itself. This goes far beyond good UI design and covers aspects such as adapting tools to specific process models and modes of usage (possibly individualized). Furthermore, the project should be capable of supporting users at different levels of expertise, from beginners to power users. This could mean that the framework/tool should adjust its interfaces and capabilities dynamically. It should also be possible to adjust the framework/tool to different languages and cultures.

Interoperability
Wherever possible and pragmatically feasible, the approach should be to support industry *de facto* and *de iure* standards (as opposed to proprietary and custom solutions). This should facilitate interworking with external toolsets as well as within the project itself and will also take advantage of the greater exposure that standards have, in general. Also included here are (1) a list of project dependencies for the project and (2) a list of standards supported by the project.

Projects at Eclipse

Overview

The Eclipse modeling community has created a large number (60 and counting) of open source projects that support different aspects of model-based engineering. Many of these projects, such as the Eclipse Modeling Framework (EMF) and Graphical Modeling Framework (GMF), have been widely adopted by software vendors and corporate IT departments. However, until very recently there has not been a coordinated open source effort to create a unified modeling platform that serves as the foundation to enable a tool chain for model-based development across the life-cycle.

Projects

Here we consider Eclipse projects under the modeling umbrella, that support each of the capabilities above in some way (a summary of which projects support which capabilities can be found in Appendix A). For each capability, we identify which projects might be suitable as part of a tooling solution, and make recommendations, based on their evaluations against the criteria described earlier. *However, it should be noted that some of these recommendations are based on formal criteria using publically available data and, therefore, may not accurately reflect actual user experience and opinion. The limited time and resources available for conducting this study and the sheer multitude of projects precluded a deeper investigation of the latter aspects, although these may, in some cases, prove to be contrary to what the formal criteria suggest. Consequently, the recommendations provided herein should be interpreted with discretion, requiring more thorough study prior to any commitments.*

A summary of how well, in our estimation (based on information gathered from readily available sources, input provided by project leads, inside knowledge of the motivations behind the companies involved in the projects, and experience using the frameworks/tools in developing proprietary tools) can be found in Appendix B. Detailed information collected for each project is included in Appendix C.

Authoring

The **Eclipse Modeling Framework Technology (EMFT) Ecore Tools**, **Model Devlopment Tools (MDT) Metamodel Specification Tools (MST)**, **MDT Papyrus**, and **MDT Unified Modeling Language 2.x (UML2) Tools** projects provide end-user tooling for authoring models based on industry standard metamodels (i.e., MOF and UML).

The EMFT Ecore Tools project provides a complete environment to create, edit and maintain Ecore models (which are isomorphic to Essential MOF, or EMOF, models). It eases handling of Ecore models with a Graphical Ecore Editor and bridges to other existing EMF tools (Validation, Search, Compare, EMFatic, generators, etc.). The Graphical Ecore Editor implements multi-diagram support, a custom tabbed properties view, validation feedback, and refactoring capabilities. The long-term goal is to provide the same level of services as does JDT for Java.

The MDT MST project aims to provide tooling for the development of Meta-Object Facility (MOF) compliant metamodels and specifications based on them. One of its goals is to customize and/or extend the existing (or forthcoming) UML editors (primarily for class and package/profile diagrams) to expose Complete MOF (CMOF) concepts which are missing in UML (like identifiers, namespace URIs, and XMI tags).

The MDT Papyrus project is a graphical editing tool that provides UML2 editors and an environment to create and combine any graphical editor (domain-specific modeling language) defined from Ecore metamodel or from a UML2 profile. It consists mainly of graphical editors that are complemented by text-based and tree-based editors. All these editors allow simultaneous viewing of multiple diagrams of a given model. This model can be UML, UML profile, or any EMF-based model and it can reference other models (from any EMF-based modeling language). Papyrus provides a main editor, showing one or several model diagrams that can be arranged in tabbed views, and additional views including a model explorer view, a properties view, and a bird's-eye view. The multiple diagrams are managed by Papyrus rather than by Eclipse. The tabbed views can be arranged side by side (left, right, top and bottom), can be created, re-sized, moved, and deleted by dragging them to the desired position. A new almost fully revised version of Papyrus is currently under development and slated for bet release in the summer of 2010.

The MDT UML2 Tools project provides a Graphical Modeling Framework editor for manipulating UML models. It includes support for structure diagrams (class, profile definition, composite structure, component, and deployment), behavior diagrams (activity, state machine, and use case) and interaction diagrams (sequence and timing), as defined in the UML specification.

The **EMFT Extended Editing Framework (EEF)**, **EMFT EMF Refactor**, and **Generative Modeling Technology (GMT) Epsilon** projects provide tools and capabilities that support model authoring.

The EMFT EEF project aims to improve the EMF model creation process by providing services dedicated to editing that are more appealing than the traditional EMF editing capabilities. These services and elements are provided via a generative approach similar to the one used by the EMF.Edit component of the EMF (Core) project. The framework provides advanced editing components for the properties of EMF elements and a default generation based on standard metamodels using these components. The generic generators create a standard architecture with advanced graphical components to edit EMF model objects. These components are meant to leverage every aspect of the Eclipse Platform for example the Eclipse Dynamic Help.

The EMFT EMF Refactor project will provide extensible tool support for generating and applying refactorings to EMF-based models, including UML models. It will consist of a refactoring generation module for specifying EMF model refactorings using several model transformation approaches, a structured suite of predefined EMF model refactorings, and a refactoring application module for applying EMF model refactorings in a uniform and user-friendly way.

The GMT Epsilon project provides a family of metamodel-agnostic languages for creating, querying and modifying EMF (and other types of) models in various ways. Apart from these languages, Epsilon also contains several smaller tools and utilities that aid in model authoring.

> EuGENia is a front-end for GMF. Its aim is to speed up the process of developing a GMF editor and lower the barriers to entry for new developers. To this end, EuGENia enables developers to generate a fully-functional GMF editor by specifying a few high-level annotations in the Ecore metamodel.
>
> Exeed is an enhanced version of the built-in EMF reflective tree-based editor that enables developers to customize the labels and icons of model elements simply by attaching annotations to the respective EClasses in the Ecore metamodel. Exeed also supports setting the values of references using drag-and-drop instead of using the combo boxes in the properties view.
>
> ModeLink is an editor consisting of 2-3 side-by-side EMF tree-based editors, and is very convenient for establishing (weaving) links between different models using drag-and-drop.
>
> Epsilon also provides a set of ANT tasks to enable developers assemble complex workflows that involve both MDE and non-MDE tasks.

Concordance is a tool that monitors selected projects of the workspace and maintains an index of cross-resource EMF references. Concordance can then use this index to automatically reconcile references when models are moved, and report broken references when models are updated/deleted.

The **Graphical Modeling Project (GMP) Graphical Modeling Framework (GMF)**, **Presentation Modeling Framework (PMF)**, and **Textual Modeling Framework (TMF) Xtext** projects provide support for creating editors to edit models using one or more concrete syntax, i.e., either graphically, textually, or both. These support model authoring via the end-user consumable tooling they generate.

The **MDT Business Process Model and Notation 2.x (BPMN2)**, **MDT UML2**, and **MDT XML Schema Definition (XSD)** projects provide reference implementations of industry standard metamodels. They provide indirect support (change notification, persistence support via XMI serialization, and a very efficient reflective API, among other things) for authoring model instances based on those standards.

The **MDT Object Constraint Language (OCL)** project provides an implementation of the OCL OMG standard for EMF-based models. It defines an Ecore implementation of the OCL abstract syntax model, including support for serialization of parsed OCL expressions, to support for the authoring of constraints on arbitrary models. MDT OCL provides a visitor API for analyzing/transforming the AST model of OCL expressions and an extensibility API for clients to customize the parsing and evaluation environments used by the parser.

Recommendations
Of the projects that support authoring, EMFT Ecore Tools, MDT Papyrus, EMFT EEF, GMT Epsilon, GMP GMF, TMF Xtext, MDT UML2, MDT XSD, and MDT OCL rate higher in terms of stability and quality (see Appendices B and C for details). However, given that EMFT Ecore Tools and MDT Papyrus are end-user tools that make use of other projects as supporting technologies, and, given that MDT MST, EMFT EMF Refactor, PMF, and MDT BPMN2 are relatively immature (whereas most of the others have been around for several years), it is perhaps too early to arrive at a true evaluation for them.

*We recommend **MDT Papyrus**, **EMFT EEF**, **GMP GMF**, **TMF Xtext**, **MDT UML2**, **MDT XSD**, and **MDT OCL** as suitable technologies to support authoring. It should be pointed out, however, that MDT Papyrus is still quite far from being ready for prime time, and would perhaps require additional attention and investment in order to be viable as an industrial strength authoring environment. EMFT Ecore Tools is not recommended because there is effectively no committer community for it at the present moment; however, it could represent a viable solution for MOF modeling without much effort. We also suggest keeping an eye on GMT Epsilon as it evolves, since it appears to have the potential of being a promising solution in the future.*

Code Generation

Both the **EMF EMF (Core)** and **MDT UML2** projects provide support for generating code that targets the EMF runtime. Note that the term “runtime” is used here in a general sense (in contrast to the notion of runtime in the context of RSD RT, for example), to refer to the core set of EMF plug-ins that are needed to run an EMF-based application.

EMF (Core) provides a code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. The EMF code generation facility is based on the Java Emitter Templates (JET) language and is capable of generating everything needed to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked. The generation facility leverages the JDT (Java Development Tooling) component of Eclipse. All generators support regeneration of code while preserving user modifications. The generators can be invoked either through the GUI or headless from a command line.

MDT UML2 provides an extended version of the code generation facility in EMF that's specialized to handle concepts that are unique to using UML models as the source model (e.g., redefinition, subsets, and union constraints) and provides some additional customizations (such as factory and look-up API methods) to make the resulting code easier to work with.

The **EMFT Eclipse Generation Factories (EGF)** project supports the integration of transformation engines (model-to-model, model-to-text, text-to-model, text-to-text) for flexible software mass-production. It federates generation around the pivotal element of factory component. A factory component is a unit of generation with a clear objective of generation. It has a contract to declare the factory component parameters. Contrary to a classic generation which simply combines all the generation parameters, generation data is organized by viewpoints, such as generation pattern, license description or deployment declaration. Altogether, viewpoints explicitly configure all the generation parameters. The generation orchestration is defined by a production plan. A generation step of the production plan can either be a Java task or, by assembly, an invocation of another factory component. Regarding the lifecycle, a factory component can be updated and re-executed.

The **EMFT Texo** project provides model and template driven development technology for web application development projects. It generates true POJOs (Plain Old Java Objects) from an Ecore model or XSD without direct compile time link from generated entities to EMF, Texo, or other framework types. Texo facilitates integration of EMF concepts with other frameworks such as Google Web Toolkit or Object Relational Mapping solutions and provides overridable and extendable code generation with support for EMF-like merge, formatting, and import-organizing.

The **GMT Epsilon**, **Model to Text (M2T) Acceleo**, **M2T Java Emitter Templates (JET)**, and **M2T Xpand** projects all provide template-based languages that support generation of arbitrary textual artifacts, including code.

The GMT Epsilon project provides a family of metamodel-agnostic languages for creating, querying and modifying EMF (and other types of) models in various ways. The Epsilon Generation Language (EGL) is a template-based model-to-text language for generating code, documentation and other textual artifacts from models. EGL supports content-destination decoupling, protected regions for mixing generated with hand-written code, and template coordination.

The M2T Acceleo project (previously named MTL) provides an implementation of the MOF Model To Text OMG standard, including support for almost all of the OMG specification keywords. One does not need to be an expert to get started using the plug-ins. The Acceleo template editor is associated with Acceleo source files (.mtl files) and provides all the features of a modern programming editor to make you more productive when developing code generation modules, including syntax highlighting, smart completion, navigation to definitions and references, error highlighting, dynamic outline, and code folding.

The M2T JET project provides an expanded version of Java Emitter Templates (JET) language (from EMF Core) to support custom tags (which are distributed in "tag libraries"). It defines Java interfaces and Eclipse extension points for declaring custom tag libraries and includes standard JET tag libraries that make it possible to create entire transformations without recourse to Java and the Eclipse APIs. M2T JET also provides Eclipse API and UI for invoking such transformations and a JET template editor (formerly a separate component, the EMFT JET Editor).

The M2T Xpand project provides a statically-typed template language featuring polymorphic template invocation, aspect oriented programming, functional extensions, a flexible type system abstraction, model transformation, model validation, and much more. It includes an editor which provides features like syntax coloring, error highlighting, navigation, refactoring, and code completion. Xpand was originally developed as part of openArchitectureWare project before it became a component under Eclipse.

The **MDT Papyrus** project is a graphical editing tool that provides UML2 editors and environment to create and combine any graphical editor (domain-specific modeling language) defined from Ecore metamodel or from a UML2 profile. One of the goals of the Papyrus project is to provide support for code generation (for Java, C/C++, and ADA) based on functionality that was originally part of Papyrus I (see http://www.papyrusuml.org).

The **PMF** project is a modeling solution and code generation facility to build enterprise data presentation applications. It provides the basic functional concepts of user interaction in a platform-independent model (PIM) level UI modeling language. This framework is highly extensible and can be integrated with any UI technology such as SWT/JFace, e4, JSF, XUL, Swing, GWT, Ajax, Silverlight or others. The final goal is to provide a rich-feature, intrusive, high extensible UI MDA framework.

The **Textual Modeling Framework (TMF) Xtext** project provides a framework for development of programming languages and domain specific languages (DSLs). Given a DSL described using Xtext's simple EBNF grammar language, the generator will create a parser, an AST-meta model (implemented in EMF), as well as a full-featured Eclipse text editor from that. As part of openArchitectureWare, Xtext integrates with EMF-based code generator frameworks and is fully supported by Xpand (see above).

Recommendations
Of the projects that support code generation, EMF EMF (Core), MDT UML2, M2T Acceleo, M2T Xpand, and TMF Xtext rate higher in terms of stability and quality (see Appendices B and C for details). EMF EMF (Core) and MDT UML2 require the EMF runtime and TMF Xtext is specialized for textual modeling.

*We recommend **M2T Acceleo** and **M2T Xpand** as suitable technologies to support code generation. It is worth noting that while M2T Xpand is by far the most customizable and extensible of the code generation technologies at Eclipse, M2T Acceleo may be a superior choice in terms of scalability and interoperability. Although M2T JET does have high predictability, customizability, and extensibility, it is not recommended because of the lack of diversity in its various communities (particularly its committers and users) and, as a result, uncertain long-term sustainability.*

Collaborative Development

The **EMF Connected Data Objects (CDO)** and **EMFT Modeling Team Framework (MTF)** both facilitate collaborative development through the Team support provided in Eclipse.

The EMF CDO project provides a Model Repository that is a distributed shared model framework for EMF models and metamodels. CDO is also a model runtime environment with a focus on orthogonal aspects like model scalability, transactionality, persistence, distribution, queries, and more. CDO has a 3-tier architecture supporting EMF-based client applications, featuring a central model repository server and leveraging different types of pluggable data storage back-ends like relational databases, object databases and file systems. The default client/server communication protocol is implemented with the Net4j Signalling Platform.

The EMFT MTF project will provide a meta repository on top of different repository types. It could become the base for software configuration management of Eclipse projects which uses the Eclipse Modeling Framework (EMF). The Modeling Team Framework will track the mapping between metamodel versions and editor plug-ins and will automatically provide p2-based update sites. MTF will define APIs and implement components, based on EMF and the Eclipse Team API, for storing, collaborating and synchronizing all software development artifacts in modeling projects. It will provide a UI for consistent versioning handling of all these artifacts and increase the usability of EMF-based modeling tools by simplifying the way editor plug-ins are loaded for specific metamodels. MTF will integrate with other Eclipse components, e.g., EMF CDO, for storing and collaborating on EMF models, or Subclipse for storing text files in SVN repositories.

Recommendations
Of the two projects that support collaborative development, EMF CDO rates higher in terms of stability and quality (see Appendices B and C for details). Given that the EMFT MTF is relatively new (whereas EMF CDO has been around for several years), it is perhaps too early to arrive at a true evaluation for it.

*We recommend **EMF CDO** as a suitable technology to support collaborative development. It has a particularly diverse committer community and is the most (perhaps the only) viable option for developing truly scalable modeling applications based on EMF. We also suggest keeping an eye on **EMFT MTF** as it evolves, since it appears to have the potential of being a promising solution in the future.*

Compare / Merge

The **EMF Compare** project provides generic model compare and merge support for any kind. The objective of this component is to provide a stable and efficient generic implementation of model comparison and to provide an extensible framework for specific needs. The comparison process used by EMF Compare is divided in 2 phases: matching and differencing. The matching phase browses model versions, figuring out which elements come from which other ones, and the differencing process browses the matching results to create the corresponding deltas which may themselves be serialized as models.

The **EMF EMF (Core)** project provides a generic change model for representing changes, or deltas, to an instance model of any other EMF model. The EMF change model can be used to represent and make changes to objects, or to record changes as they're being made, enabling the modeler to later roll them back (i.e., undo them).

As described above, the **GMT Epsilon** project provides a family of metamodel-agnostic languages for creating, querying and modifying EMF (and other types of) models in various ways. The Epsilon Comparison Language (ECL) is a metamodel agnostic rule-based language for discovering correspondences (matches) between elements of models. The Epsilon Merging Language (EML) is a metamodel agnostic rule-based language for merging models after identifying their correspondences with ECL (or otherwise).

Recommendations
Of the projects that support compare/merge, EMF EMF (Core) rates higher in terms of stability and quality (see Appendices B and C for details)..

*We recommend **EMF Compare** as a suitable technology to support compare/merge. We do caution, however, that it currently has issues with scalability,. Also, it does not provide a graphical rendering of the differences between two versions – a feature that is typically in demand for UML models.*

Configuration Management

The **EMF CDO**, **EMF Teneo**, **EMFT JCR Management (JCRM)**, and **EMFT MTF** projects provide support for configuration management of instance models.

As described above, the EMF CDO project provides a model repository that is a distributed shared model framework for EMF models and metamodels. It supports management of model artifacts to a very fine level of granularity, independently of the underlying persistence technology.

The EMF Teneo project provides a database persistency solution for EMF using Eclipselink or Hibernate. It supports automatic creation of EMF to Relational Mappings and the related database schemas. The solution contains a runtime layer to support specific EMF features. EMF Objects can be stored and retrieved using advanced queries (HQL or EJB-QL). EMF resource implementations are provided for integration with EMF Editors. The persistence logic and mapping can be controlled using

EJB3/JPA-like annotations. Most of the EJB3/JPA mapping standard is supported. Teneo is used for Model Relational mapping within the CDO Hibernate Store.

The EMFT JCR Management project will provide tooling and a Java Content Repository (JCR) persistence framework for EMF with pluggable JCR implementations. It allows combining the strength of the Eclipse modeling projects with the scalability, features, and exchangeability of the JCR repositories.

The **EMFT Edapt** project will provide an extensible framework for reducing the migration effort that results from the evolution of Ecore models. Since the number of existing instances of a successful modeling language typically outnumbers the number of editors, interpreters and transformations, instance migration effort dwarfs tool reconciliation effort. Consequently, Edapt clearly focuses on the migration of instances in the first step. However, we plan to make it extensible with respect to the migration of other artifacts like editors, interpreters and transformations. The basic idea behind Edapt is to obtain the changes between two versions of an Ecore model, and to enrich them with information about how to automatically migrate existing instances. Edapt will support a pluggable mechanism to obtain the changes between two Ecore model versions along with two examples: the operation-based and the difference-based approach.

Recommendations
Of the projects that support configuration management, EMF CDO rates higher in terms of stability and quality (see Appendices B and C for details). Given that the EMFT MTF and EMFT Edapt are relatively immature (whereas EMF CDO has been around for several years), it is perhaps too early to arrive at a true evaluation for them.

*We recommend **EMF CDO** as a suitable technology to support configuration management. In contrast with a traditional SCM system like ClearCase, a model repository like CDO provides benefits such as object-level versioning, live synchronization, and semantics-aware compare and merge. We also suggest keeping an eye on EMFT MTF as it evolves, since it appears to have the potential of being a promising solution in the future.*

Debug & Trace

None of the Modeling projects at Eclipse currently provides support for debug and trace capabilities. While the EMFT Model Execution Framework (MXF) project (see below) describes itself as providing facilities for debugging and tracing the runtime execution of models, these are done via a reflective interpreter rather than execution on a real target.

Recommendations
We recommend working with the open source community to cultivate tooling that supports debug and trace capabilities or, perhaps less optimally, to seek a proprietary alternative.

DSL Support

The **EMF EMF (Core)**, **EMFT EMF Feature Model**, and **MDT XSD** projects provide support for defining metamodels that describe arbitrary Domain Specific Languages (DSLs).

The EMF (Core) project includes a meta model (Ecore) for describing domain-specific models and runtime support for those models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.

The EMF Feature Model project will define a standard representation of Feature Models inside the Eclipse platform. The intent is to provide a uniform representation for variability information for tools based on the Eclipse Modeling Framework. This will allow easy and consistent access to variability-related information, such as variation points and variant decisions, in DSLs, M2M transformations, and other contexts where variability information is produced or consumed. EMF Feature Model is the only Eclipse Modeling project that provides explicit support for the notion of product families.

The MDT XSD project provides an API for manipulating the components of an XML Schema as described by the W3C XML Schema specifications, as well as an API for manipulating the DOM-accessible representation of XML Schema as a series of XML documents, and for keeping these representations in agreement as schemas are modified. The library includes services to serialize and deserialize XML Schema documents, and to do integrity checking of schemas (for example, not using a maximum value for a simpleType which is invalid considering the base type of that simpleType). The project goal is to support 100% of the functionality of XML schema representation, but not necessarily to provide documentation against schema assessment or validation services, which are normally provided by a validating parser, such as Apache's Xerces-J.

The **GMP GMF** project provides a generative component and runtime infrastructure for developing graphical editors for arbitrary model languages based on EMF and the Graphical Editing Framework (GEF). The project aims to provide these components, in addition to exemplary tools for select domain models which illustrate its capabilities.

The **MDT MST**, **MDT Papyrus**, **MDT UML2**, and **MDT UML2 Tools** projects support the definition of domain specific languages through the application of UML profiles.

The MDT MST project aims to provide tooling for the development of Meta-Object Facility (MOF) compliant metamodels and specifications based on them. One of the goals of the project is to provide a proof of concept for changes and/or extensions to EMF and/or UML2 to better support richer and evolving metamodels, as per the MOF Support for Semantic Structures RFP.

As described above, the MDT Papyrus project is a graphical editing tool that provides UML2 editors and environment to create and combine any graphical editor (domain-specific modeling language) defined from Ecore metamodel or from a UML2 profile. In accordance with its primary goal of realizing the UML2 specification, Papyrus delivers all UML diagram editors and provides extensive support for UML profiles. It includes all the facilities for defining and applying UML profiles in a very rich and efficient manner. It provides the same powerful tool customization capabilities for both profile-based editors and DSML-like (Domain Specific Modeling Language) editors. The main difference here is to enable the profile plug-in of Papyrus to dynamically drive its customization as defined by a profile. This means that when applying a profile, the tool may adapt its visual rendering and GUI to fit the specific domain supported by that profile. Of course, if the profile is unapplied later, the tool returns to its previous configuration.

The MDT UML2 project provides an EMF-based implementation of the Unified Modeling Language (UML) 2.x OMG metamodel for the Eclipse platform. It includes complete support for the profile and stereotype mechanisms as defined in the UML specification.

The MDT UML2 Tools project provides a set of GMF-based editors for viewing and editing UML models; it is focused on (eventual) automatic generation of editors for all UML diagram types. UML2 Tools supports the creation of profile definition diagrams which facilitate the definition, registration, and application of profiles and stereotypes to augment UML to meet the requirements of a particular domain.

As described above, the **PMF** project is a modeling solution and code generation facility to build enterprise data presentation applications. It supports DSLs by providing the basic functional concepts of user interaction in a platform independent modeling language to support the development of application user interfaces.

As described above, the **TMF Xtext** project provides a framework for development of programming languages and domain specific languages (DSLs). The Framework integrates with technology from Eclipse Modeling such as EMF, GMF, M2T and parts of EMFT. Development with Xtext is optimized for short turnarounds, so that adding new features to an existing DSL is a matter of minutes. However, sophisticated programming languages can also be implemented.

Recommendations
Of the projects that provide DSL support, EMF EMF (Core), MDT XSD, GMP GMF, MDT Papyrus, MDT UML2, and TMF Xtext rate higher in terms of stability and quality (see Appendices B and C for details). MDT Papyrus is an end-user tool that uses several of the projects as supporting technologies. Also, given that EMFT EMF Feature Model, MDT MST, and PMF are relatively immature (whereas most of the others have been around for several years), it is perhaps too early to arrive at a true evaluation for them.

*We recommend **EMF EMF (Core)**, **MDT XSD**, **GMP GMF**, **MDT Papyrus**, **MDT UML2**, and **TMF Xtext** as suitable technologies to provide DSL support. It should be pointed out, however, that MDT Papyrus is still quite far from being ready for use in a large project, and would perhaps require additional attention and investment in order to be viable as an industrial strength DSL environment. We also question the sustainability of GMF on the long term, given that its committer community is in recession, and especially in light of recently proposed alternatives such as Graphiti (see the section below on proposals for details).*

Review

None of the Modeling projects at Eclipse currently provides support for review capabilities. There is, however, relevant work happening within the Mylyn project which could perhaps be leveraged within a modeling context.

Recommendations
We recommend working with the open source community to cultivate tooling that supports review capabilities or, perhaps less optimally, to seek a proprietary alternative.

Simulation

The **EMFT Model Execution Framework (MXF)** project will build extensible frameworks and exemplary tools for executable models, e.g., an editor to define metamodels and runtime models and their behavior in terms of an action language. MXF will provide integration with GMF for building DSL simulators that are seamlessly integrated into a generated editor. A prototypical implementation of the framework has been developed as the M3Actions framework and is currently available at sourceforge.org (see the project's website for further information). This framework is centered on a graphical action language to precisely define models with their executable behavior. A reflective interpreter is able to execute and debug these definitions. The overall framework supports multiple meta-layers via an explicit instantiation concept where runtime models are defined as logical instances of the abstract syntax. These runtime models are the model parts that change over time and these changes can be recorded during execution as a trace.

Recommendations
Given that EMFT MXF is relatively immature, it is perhaps too early to arrive at a true evaluation for it.

We cannot currently recommend a suitable technology to support simulation. We do, however, suggest keeping an eye on EMFT MXF as it evolves, since it appears to have the potential of being a promising solution in the future. Alternatively, we recommend working with the open source community to cultivate tooling that supports simulation capabilities or, perhaps less optimally, to seek a proprietary alternative.

Testing

As described above, the **EMFT MXF** project will build extensible frameworks and exemplary tools for executable models, e.g., an editor to define metamodels and runtime models and their behavior in terms of an action language. The common execution infrastructure will define common concepts on top of the Eclipse debugging framework and will enable applications to share runtime models, adapters for specific editors and debuggers, tracing capabilities, and more. An interpreter and debugger will support the execution and testing of models as well as recording of simulation runs for further analysis.

Recommendations
Given that EMFT MXF is relatively immature, it is perhaps too early to arrive at a true evaluation for it.

We cannot currently recommend a suitable technology to support testing. We do, however, suggest keeping an eye on EMFT MXF as it evolves, since it appears to have the potential of being a promising solution in the future. Alternatively, we recommend working with the open source community to cultivate tooling that supports testing capabilities or, perhaps less optimally, to seek a proprietary alternative.

Traceability

As described above, the **EMF EMF (Core)** project provides a very efficient reflective API for manipulating EMF objects generically. While it does not support high level services for managing trace relationships, this basic API provides the necessary building block build support into virtually any EMF-based modeling application.

As described above, the **TMF Xtext** project provides a framework for development of programming languages and domain specific languages (DSLs). When loading textual models Xtext establishes cross-links where needed. By doing so, objects build an interconnected domain model.

Recommendations
Both of the projects that provide support for traceability rate highly in terms of stability and quality (see Appendices B and C for details). However, given the nature of this support, which is limited and indirect at best.

We cannot currently recommend a suitable technology to support traceability. We recommend working with the open source community to cultivate tooling that supports traceability capabilities or, perhaps less optimally, to seek a proprietary alternative.

UAL Support

None of the Modeling projects at Eclipse currently provides support for UML Action Language (recently proposed to be renamed "Alf"). However, various members of the MDT UML2 project community have, from time to time, expressed interest in contributing to such a capability as part of the project. Furthermore, given that the basic level of the proposed standard for this language is, in effect, a subset of Java, it is highly likely that existing Eclipse-based Java development tools can be adapted to support this capability.

Recommendations
We recommend working with the open source community to cultivate tooling that support for UAL capabilities or, perhaps less optimally, to seek a proprietary alternative.

Assets

The **GMT Model Discovery (MoDisco)** project provides an extensible framework to develop model-based tools to support modernizing existing software . It provides metamodels to describe existing systems, discoverers to automatically create models of these systems, generic tools to understand and transform complex models created out of existing systems, and use-cases illustrating how MoDisco can support the modernization process. The MoDisco project was initiated and originally developed in the context of the IST European MODELPLEX project.

Recommendations
Despite its status as an incubating project, GMT MoDisco rates fairly highly *in terms of stability and quality (see Appendices B and C for details). However, the project is focused more on discovery of (legacy) assets than the management and reuse of assets.*

We cannot currently recommend a suitable technology to support assets. We do, however, suggest considering GMT MoDisco as a potentially promising solution. Alternatively, we recommend working with the open source community to cultivate tooling that supports asset capabilities or, perhaps less optimally, to seek a proprietary alternative.

Document Generation

As described above, the **GMT Epsilon**, **M2T Acceleo**, **M2T JET**, and **M2T Xpand** projects all provide template languages that support generation of arbitrary textual artifacts. In a sense, this capability applies equally well to the generation of documentation.

The **MDT MST** and **MDT Papyrus** projects aim to provide support for the generation of documentation based on industry standard metamodels, namely CMOF and UML.

As described above, the MDT MST project aims to provide tooling for the development of Meta-Object Facility (MOF) compliant metamodels and specifications based on them. One of its goals is to provide a mechanism for generating a specification document (or at least a decent boiler plate for one) directly from a metamodel by leveraging, for example, the Business Intelligence and Reporting Tools (BIRT) project at Eclipse.

As described above, the MDT Papyrus project is a graphical editing tool that provides UML2 editors and environment to create and combine any graphical editor (domain-specific modeling language) defined from Ecore metamodel or from a UML2 profile. One of the goals of the Papyrus project is to provide support for document generation based on functionality that was originally part of TOPCASED (see the section on other tools for more information on TOPCASED).

Recommendations
Of the projects that support documentation generation, M2T Acceleo, M2T Xpand, and MDT Papyrus rate higher in terms of stability and quality (see Appendices B and C for details). Given that MDT MST is relatively immature (whereas most of the others have been around for several years), it is perhaps too early to arrive at a true evaluation for it.

We cannot currently recommend a suitable technology to support document generation. We do, however, suggest considering M2T Acceleo, M2T Xpand, and MDT Papyrus as a potentially promising solutions. Alternatively, we recommend working with the open source community to cultivate tooling that supports documentation generation capabilities or, perhaps less optimally, to seek a proprietary alternative.

Reverse Engineering

The **EMFT Documentation to Model (Doc2Model)** project will provide an extensible framework for producing EMF model instances from plain text and structured documents. Doc2Model can be used to, for example, to import requirements from text files and transforming them into SysML requirements models. The documents file formats which will be managed by Doc2Model include open formats as docx, xlsx, odt, odf; common formats such as csv; and formats desired by the eclipse community. The Doc2Model API will provide extension mechanism to allow users to add custom parsers for specific tools.

As described above, the **GMT MoDisco** project provides an extensible framework to develop model-driven tools to support use-cases of existing software modernization. It supports extraction of information from an existing system to help understand aspects of this system (structure, behavior, persistence, data-flow, change impact, etc).

As described above, the **MDT Papyrus** project is a graphical editing tool that provides UML2 editors and environment to create and combine any graphical editor (domain-specific modeling language) defined from Ecore metamodel or from a UML2 profile. One of the goals of the Papyrus project is to provide support for reverse engineering based on functionality that was originally part of TOPCASED (see the section on other tools for more information on TOPCASED).

Recommendations
Of the projects that support reverse engineering, GMT MoDisco and MDT Papyrus rate higher in terms of stability and quality (see Appendices B and C for details). EMFT Doc2Model is relatively immature (whereas most of the others have been around for several years), it is perhaps too early to arrive at a true evaluation for it.

We cannot currently recommend a suitable technology to support reverse engineering. We do, however, suggest considering GMT MoDisco and MDT Papyrus as a potentially promising solutions. Alternatively, we recommend working with the open source community to cultivate tooling that supports reverse engineering capabilities or, perhaps less optimally, to seek a proprietary alternative.

Transformation

The **EMFT Henshin** project provides an in-place model transformation language for the Eclipse Modeling Framework. The framework supports direct transformations of EMF model instances (endogenous transformations), as well as generating instances of a target language from given instances of a source language (exogenous transformations). The main language features include support for endogenous as well as exogenous transformations, natural treatment and efficient in-place execution of endogenous transformations, graphical syntax, and support for static analysis of transformations.

The **GMT Epsilon** project provides a family of metamodel-agnostic languages for creating, querying and modifying EMF (and other types of) models in various ways. The Epsilon Transformation Language (ETL) is a rule-based model-to-model transformation language that supports transforming many input to many output models, rule inheritance, lazy and greedy rules, and the ability to query and modify both input and output models. The Epsilon Wizard Language (EWL) is a language tailored to interactive in-place model transformations on model elements selected by the user. EWL is integrated with EMF/GMF and as such, wizards can be executed from within EMF and GMF editors.

As described above, the **GMT MoDisco** project provides an extensible framework to develop model-driven tools to support use-cases of existing software modernization. It supports transformation of an existing system to integrate better coding norms or design patterns and also to change a component, the framework, the language, or its architecture.

The **GMT UMLX**, **Model to Model (M2M) ATLAS Transformation Language (ATL)**, **M2M Declarative Query View Transform (QVTd)**, and **M2M Operational Query View Transform (QVTo)** projects provide concrete syntactical support for model to model transformations.

The GMT UMLX project provides a concrete graphical syntax to complement the OMG QVT model transformation language. UMLX will conform to at least one of the QVT Export/XMI-Export Core/Relations conformance points, so that UMLX originated transformations can be used in any QVT execution environment. Much of UMLX corresponds closely to QVT relational, so the first releases will subset UMLX towards the shared capabilities. The more extensive UMLX concepts, particularly regarding multiplicity will require careful consideration of whether they can sensibly extend QVT relational, or whether it is necessary to go direct to QVT core, or to prototype an extended QVT relational for consideration by a future QVT. Conversely, QVT relational and possibly QVT core concepts that are useful may merit extension of UMLX.

The M2M ATL project provides model transformation language and toolkit to facilitate the production of a set of target models from a set of source models in the field of Model-Driven Engineering (MDE). Developed on top of the Eclipse platform, the ATL

Integrated Development Environment (IDE) provides a number of standard development tools (syntax highlighting, debugger, etc.) that aim to ease development of ATL transformations. The ATL project includes also a library of reusable transformations.

The M2M QVTd component aims to provide a complete Eclipse based IDE for the Core (QVTc) and Relations (QVTr) Languages defined by the OMG QVT Relations (QVTR) language. This goal includes all development components necessary for development of QVTc and QVTr programs and APIs to facilitate extension and reuse.

The M2M QVTo project provides an implementation of the QVT Operational Mapping language defined by the OMG Query/View/Transformation Specification (QVT). It includes a semantically sensitive editor for QVT Operational language, a QVT Operational project nature and builder, support for black-box libraries (Java code accessible from QVT), transformation invocation Java API, and an execution engine & debugger.

The **M2T Xpand** project provides a statically typed template language featuring polymorphic template invocation, aspect oriented programming, functional extensions, a flexible type system abstraction, model transformation, model validation, and much more. It includes an editor which provides features like syntax coloring, error highlighting, navigation, refactoring, and code completion. Xpand was originally developed as part of openArchitectureWare project before it became a component under eclipse.

Recommendations
Of the projects that support transformation, GMT Epsilon, GMT MoDisco, M2M and M2T Xpand rate higher in terms of stability and quality (see Appendices B and C for details). However, given the incubation status of GMT Epsilon and EMFT MoDisco, and given that M2T Xpand is specialized for model to text transformations. Also, given that EMFT Henshin is relatively immature (whereas most of the others have been around for several years), it is perhaps too early to arrive at a true evaluation for it.

*We recommend **M2M ATL** as a suitable technology to support transformation. (Note, however, that in discussions with some users of M2M ATL, concerns were expressed about the usability and scalability, to the point that some users have abandoned it. Unfortunately, it was not possible to do a further investigation of the validity of these claims, given the limited time available for this study.) We also suggest keeping an eye on EMFT Henshin and GMT Epsilon as they evolve, since they appear to have the potential of being promising solutions in the future.*

Validation

The **EMF EMF (Core)** and **EMF Validation** projects provide low level support for the definition and execution of validation rules on EMF-based models.

The EMF EMF (Core) project provides a mechanism for defining what constitutes a valid state for objects modeled in EMF. It supports the declaration of invariants and constraints on EMF-based implementations, using annotated Ecore models, which can be evaluated statically via generated code or dynamically via EMF reflection.

The EMF Validation project provides an API for defining constraints for any EMF meta-model (batch and live constraints). It includes an extensibility API to support meta-models that require custom strategies for model traversal, provides support for parsing the content of constraint elements defined in specific languages and support for two languages: Java and OCL. The project also provides API support to define "client contexts" that describe the objects that need to be validated and to bind them to constraints that need to be enforced on these objects, and support for listening to validation events.

The **GMT Epsilon** project provides a family of metamodel-agnostic languages for creating, querying and modifying EMF (and other types of) models in various ways. The Epsilon Validation Language (EVL) is a model validation language that supports both intra and inter-model consistency checking, constraint dependency management and specifying fixes that users can invoke to repair identified inconsistencies. EVL is integrated with EMF/GMF and as such, EVL constraints can be evaluated from within EMF/GMF editors and generate error markers for failed constraints.

The **M2T Xpand** project provides a statically-typed template language featuring polymorphic template invocation, aspect oriented programming, functional extensions, a flexible type system abstraction, model transformation, model validation, and much more. It includes an editor which provides features like syntax coloring, error highlighting, navigation, refactoring, and code completion. Xpand was originally developed as part of openArchitectureWare project before it became a component under eclipse.

As described above, the **MDT Object Constraint Language (OCL)** project provides an implementation of the OCL OMG standard for EMF-based models. It supports model validation via its APIs for parsing and evaluating OCL constraints and queries on EMF-based models.

Recommendations
All of the projects that support rate fairly high in terms of stability and quality (see Appendices B and C for details). However, given the incubation status of GMT Epsilon, and given that M2T Xpand is specialized for model to text transformations, EMF EMF (Core), EMF Validation, and MDT OCL are not.

*We recommend **EMF EMF (Core)**, **EMF Validation**, and **MDT OCL** as suitable technologies to support validation. We also suggest keeping an eye on GMT Epsilon as it evolves, since it appears to have the potential of being a promising solution in the future.*

Proposals

With a current total of sixty subprojects, the Eclipse Modeling project has set a precedent for rapid and continued growth, and there are no signs of this trend slowing down in the near future. Indeed, a significant number of the new project proposals received by the Eclipse Foundation fall underneath the Modeling umbrella. This section covers a few of the recent proposals that are particularly relevant to this study. The relative position of these projects with respect to the dependency architecture shown previously is depicted in the diagram at the end of this section.

Graphiti

The Graphiti framework utilizes Eclipse's GEF and Draw2D for diagramming and supports EMF on the domain side. The diagrams are described by a platform independent metamodel and the diagram data is kept strictly separate from the domain data. This enables rendering an existing diagram in various environments (besides Eclipse this could be e.g. within a browser) even without having access to the underlying domain data. Graphiti is strictly API-centric: a user of the framework writes plain Java coding and (besides Graphiti) only needs to know EMF to use the framework and to build an editor - no knowledge of Draw2D or GEF is required. Graphiti complements the offering of GMF which follows a generative approach while Graphiti follows a runtime-oriented and API-based approach.

Editors built with the framework are equipped with a standardized look and feel which leads to a more coherent UI in Eclipse-based tools. Rapid prototyping is supported by simple APIs and base classes which can be used to refine an editor in an evolutionary way. The user of the framework writes so-called features to add functionality to the editor. For instance, one would write features for creating new model objects and their graphical representation, or for creating a graphical representation for an already existing model object. The complete life cycle (creating, editing, renaming, moving, deleting...) of model objects and their graphical representations can be defined and controlled by implementing such features. The standard behavior for the different operations is covered by so-called default features, if the user of the framework does not declare any special behavior.

Step by step, these default features can be replaced or extended with further functionality. Additionally, the framework comes with a hook for so-called custom features to implement non-standard behavior and further operations inside the tool.

The initial code contribution will come from SAP where Graphiti was developed originally as part of their MOF based modeling infrastructure. Later Graphiti was ported to EMF; this version will be the basis for the coding contributed to Eclipse.

Recommendations
We recommend that Graphiti be considered as a potential future alternative to the GMP GMF project. The fact that it already in widespread use by a company (SAP) which is increasing its involvement in Eclipse, whereas the companies involved in GMF (IBM, Borland) are decreasing theirs, suggests that it may in fact represent a more viable long term foundation for DSL support and model authoring capabilities.

Sphinx

Sphinx is a project providing a modeling tool platform for Eclipse that eases the development of IDE-like tool support for modeling languages used in software and systems development.

The objectives of the Sphinx project are:

> To provide an open and extensible platform enabling rapid creation of integrated modeling environments (IME) for individual or multiple modeling languages;
>
> To ensure that the resulting modeling tool support yields industrial strength scalability and robustness out-of-the box;
>
> To support a domain- and vendor-independent interoperability layer (backbone) for off-the-shelf and in-house modeling tool components supporting the same or different modeling languages which users can easily combine to create individually tailored continuous modeling tool chains;
>
> To support controlled and coordinated management either of individual modeling tool components (e.g., update, upgrade, addition, removal, activation, deactivation, etc.) or of entire modeling tool chains (e.g., coordinated switchover to new versions of multiple mutually coupled modeling tool components, reverting to old modeling tool chain configurations in case of problems or when needing to work on retired projects).

Sphinx does not aim at providing finished integrated modeling tool environments for any specific modeling language. Instead, it will be focused on common infrastructure that is typically required when developing such tool environments. This infrastructure will all the same be of strictly generic nature and in no way depend on the modeling language(s) to be supported.

It is also important to note that the intention of Sphinx is not to reinvent or replace any existing parts of Eclipse, in particular the various components of the Eclipse Modeling Project (EMP). Sphinx will rather leverage many of those and build upon them. The mission will be to provide the glue and additional things that are necessary to orchestrate these components in a consistent manner and to realize accustomed integrated tool environments supporting one or several modeling languages at reasonable effort and cost.

A good part of the ideas and code contributions behind this modeling tool platform originate from the EDONA research project, funded by the French government.

Recommendations
We recommend considering Sphinx as a strong technology candidate to facilitate collaborative development, DSL support, traceability, authoring, and validation, among other target criteria. The platform upon which the initial contribution for this

project is based (ARTOP) is already proven to be somewhat successful. It is clear that Sphinx already provides a solution to many of the requirements identified by the Eclipse Modeling Platform Working Group (see below).

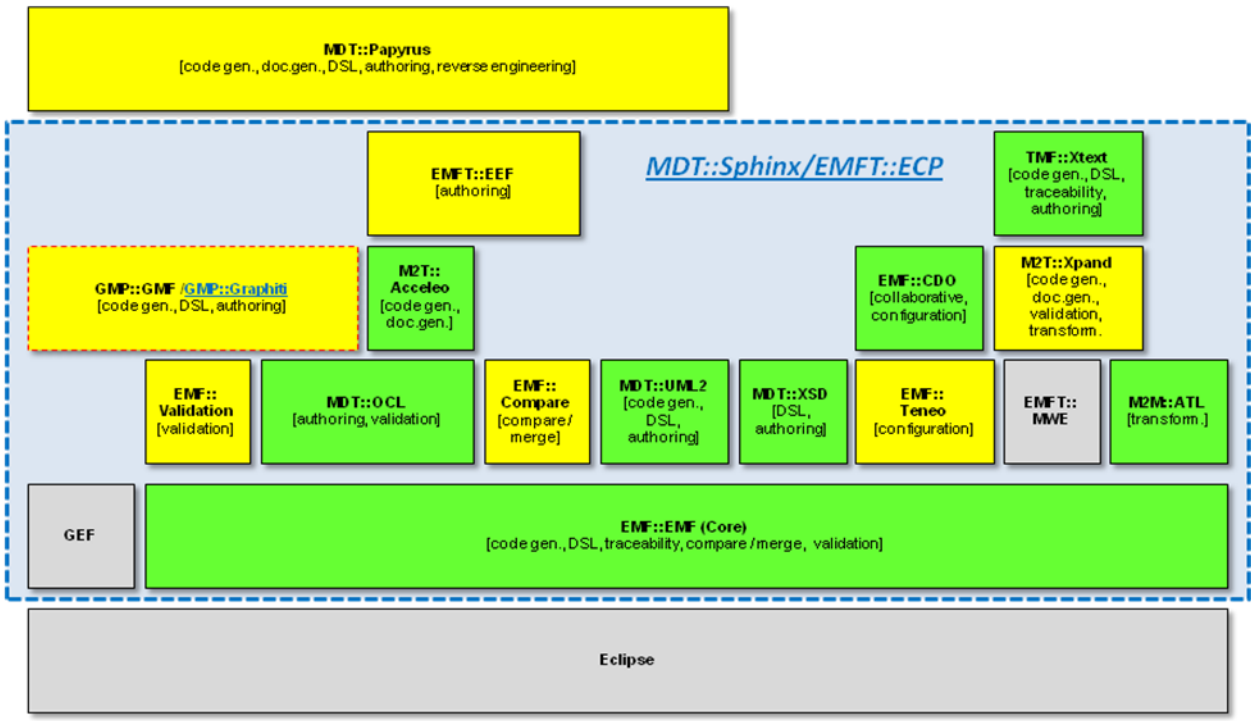
EMF Client Platform (ECP)

The goal of the EMF Client Platform (ECP) is to provide an entity-centric client application based on an EMF (Ecore) model. The only initial step required to create an application is to generate the model itself. All additional components are either generic or adapted reflectively. ECP provides a reflective UI for entity-centric applications. This includes a tree-based navigator, a form-based editor for single entities and a view for EMF validation results. These UI elements do not have to be generated, but work with an EMF model from scratch. Furthermore, ECP provides a transparent model workspace in which to store models. The developer does not have to deal with files or resources, but can access any EObject using EMF-based APIs. The models in a workspace can be shared among different clients via a model repository.

Several components of the proposed ECP project have already been developed as part of the existing UNICASE project.

Recommendations

We recommend considering keeping an eye on ECP to see whether it gains any traction within the Eclipse community. The EMFStore technology its uses (as a repository) has already started to attract attention (e.g., the redView project recently abandoned EMF CDO in favor of it), and it stands to reason that ECP may follow suit. If it does find success, ECP may represent a viable technology choice for supporting collaborative development.



Mylyn Restructuring

Although application lifecycle management (ALM) is not covered in the tool capabilities considered by this study, it is worth mentioning the Mylyn project and its planned evolution. Mylyn is the ALM framework for Eclipse that provides.

- A task management tool for developers
- A broad ecosystem of Agile and ALM integrations
- A task-focused interface

Due to the growing adoption of Mylyn's frameworks, there are now plans to restructure the project around its API boundaries for its key components such as Tasks and Context. The goal of the proposed sub-projects is to address the growth of Mylyn by creating sub-projects for the components, and to solicit additional participation from ISVs and other stakeholders in the new sub-projects.

Recommendations
We recommend considering the restructured Mylyn as a strong technology candidate for ALM functions that provides a significant portion of the functionality currently offered by IBM's Jazz framework.

Other Tools

Overview

In addition to projects formally endorsed by the Eclipse Foundation, there is a perhaps surprisingly large number of open source tools available for model-based engineering. In this study, we have investigated more than 50 model-oriented tooling projects, uncovered through relatively thorough search of the internet (especially the open source sites), as well as through discussions with various experts.

Note that many of the items considered here are also built on the Eclipse platform, but are not official Eclipse projects. Some of these projects stand a good chance of being formally adopted by the Eclipse Foundation as projects in their own right or being integrated into one of the existing projects. However, we also felt it useful to do at least an initial investigation of projects outside the Eclipse framework, especially given that some of them have a long and relatively successful track record and a significant user base that may be approaching critical mass. In addition, we have also studied and included a number of well-established research efforts that could be the potential source of significant new capabilities.

Methodology

Most of the information on non-Eclipse projects was gleaned from a thorough search of Internet-based sources (primarily project web pages, user forums, and open source repositories), supplemented in several cases with information from direct discussions or e-mail correspondences with users and developers, as well as from personal knowledge and experiences. Wherever possible, multiple sources were consulted on each individual effort and the results cross-referenced and correlated. However, in a small number of cases, only single sources were available.

DISCLAIMER: *Although significant effort has been put into verifying the information contained in this section and accompanying tables in the corresponding appendices, given the sheer diversity and wide dispersal of information sources as well as the limited time and resources available, it is not possible to fully guarantee either its completeness or its accuracy.*

Tools

As is the case with the formally endorsed Eclipse projects, there is significant fragmentation here as well—perhaps even more pronounced. Most of the tools are actually authoring tools (primarily UML 2 tools), but many of them incorporate other capabilities and a few of them are intended to complement authoring tools.

Perhaps because they were not developed under the auspices of a motivating larger host project, many of these projects are small-scale efforts, sometimes based on the work of one or two enthusiastic individuals. Consequently, a large proportion of them are either defunct or nearly defunct, while only a few have the scope, capabilities, and robustness required for an industrial strength tool. Others are based on highly-specialized environments Nevertheless, with the exception of a few cases that seem to lead to dead ends (e.g., no references or detectable activity in the past 3-4 years), in the tables included in the appendices we list all the model-based tools encountered during our search, their capabilities, along with accompanying evaluations of them. To avoid an excess of irrelevant or less relevant information, in this section we will only expand on some of the. The relevance assessment for all tools investigated can gleaned from the "Relevance" column in the table in Appendix E, which also includes an explanation for the assigned rating.

AndroMDA (pronounced "Andromeda") is an extensible code generation framework that XMI inputs from several commercial UML tools and produces corresponding code components for a variety of different languages and runtime environments (e.g., Java/J2EE, Hibernate, EJB, Spring, etc.). A significant number of pre-defined translation "cartridges" already exist for the most popular code generators, but new ones can be constructed. What makes it potentially interesting, in addition to its claimed flexibility, is that it has been used on a number of commercial projects and that it has what appears to be a vibrant and active developer community.

BIRT (Business Intelligence and Reporting Tool) is an Eclipse-based open source reporting system for web applications, especially those based on Java and J2EE. In contrast to the rest of the projects in this section, it is actually an official Eclipse project, but sits outside the general modeling space. BIRT has two main components: a report designer based on Eclipse, and a runtime component that can be added to an application server. BIRT also offers a charting engine that enables adding charts to an application. It appears to be the most effective and, hence, most widely used documentation generation facility in Eclipse.

BOUML is a relatively comprehensive UML authoring package with code generators for a number of output languages (Java, C++, PHP, Python, IDL), primarily intended to be used in round-trip fashion. It has some support for collaborative development and configuration management. What makes it interesting is that it seems to scale up very well (much better than most commercial UML tools, it is claimed (http://bouml.free.fr/benchmark.html)) and has been used in industry (although no references are provided). There is also an adopter community around it that has produced additional capabilities to be used with the tool. Unfortunately, the main authoring and code generation capabilities seem to be one-person development efforts.

Coral is a metamodel-independent toolkit that is part of the greater MDE research project conducted at the Åbo Academy (Finland). It can be used to create, edit and transform new models and modeling languages at run-time. Coral is a full metamodeling tool: it includes a modeling language, including its abstract and concrete syntax, defined by a model. It is also a toolkit and can be customized to build other modeling tools. It is being used as a platform to build tools for invariant-base programming using a formal visual notation, model-checking UML behavioral models, generation of code from UML models, and integration of hardware components in network-on-chip architectures. A number of important contributions to model management have emanated from work with Coral. Although it does not appear to have been used extensively in industry, what makes it interesting is its convenience as an experimental testbed for innovative ideas in model-based development.

Fujaba is interesting for similar reasons as Coral: it provides a convenient testbed for experimentation, especially considering its Eclipse-based and OMG focused approach. It was jointly developed by several universities in Germany involved in model-based engineering research. The **Fujaba Tool Suite** is an Eclipse-based open source tool providing developers with support for model-based software engineering and re-engineering. The Fujaba project aims at developing and extending the Fujaba Tool Suite and thus offering an extensible platform for software engineering researchers. The Fujaba Development Group is permanently developing and extending Fujaba and numerous related tools.

Fujaba's main features are:

- Powerful, easy to use, yet formal, graphical, object-oriented software system specification language (UML class diagrams and specialized activity diagrams, so called Story Diagrams based on graph transformations)

- Java code generation based on the formal specification of a systems' structure and behavior that results in an executable system prototype.

- An extensible CASE tool framework for researchers, who can develop their own Fujaba plug-ins.
- Numerous Fujaba plug-ins providing support for example for
  - Reverse engineering of source code by creating UML class diagrams, detecting design patterns, idioms, anti patterns, bad smells,...
  - Model-to-model transformations specified by triple graph grammars, TGGs (also usable in case of model synchronization)
  - Modelling, validation and verification of embedded real-time systems
  - Meta-Modelling with MOF (OMG meta object facility)

HOL-OCL is a an interactive environment for OCL that is embedded within the Isabelle theorem prover providing the opportunity for experimenting with various sophisticated formal analyses methods. However, it is not based on Eclipse and is implemented a not-so-common language, Standard ML.

Kermeta is another Eclipse-based project developed at INRIA (but in the Rennes lab), which has recently been incorporated into the TOPCASED project. Although primarily a research tool, Kermeta includes a number of interesting capabilities for code generation and simulation. It is based on a custom metamodeling language (Kermeta), and can be used to define DSLs and automatically generate corresponding tools for those languages, including, notably, simulators/interpreters. Thus, it could serve as a basis for a UML or UML-RT simulator.

MOCASEngine is a Java UML state machine engine library for executing embedded UML state machine models. MOCASEngine works with Eclipse EMF. It has a full support of composite/orthogonal states, completion transitions, guard, hierarchy of signals. What may be interesting is to use this as a base for a simulation/debug engine. However, it seems to be an effort that is in its infancy.

MOFScript is a general Eclipse-based tool for translating EMF-based models into text. It is primarily used for code generation but can, in principle, be used to translate to other models, albeit in text form (e.g., XMI). It has been used in industrial applications, but it is hard to find published evidence of its success or failure.

TOPCASED is an industry-led effort to provide an Eclipse-based open source modeling toolset for real-time and safety-critical systems. Although not an officially endorsed Eclipse project, it follows the Eclipse release schedule and a number of its tool components, such as the primary UML 2 editor, Papyrus, are or have become a part of the Eclipse MDT project. An important aspect of TOPCASED is that it is backed by some major industrial players such as EADS, Airbus, Continental, Rockwell-Collins, and Thales, all of whom have invested significant funds and resources into its development, with intent to obtain a truly industrial-strength open source toolset out of it. As such, it aligns nicely with the proposed Eclipse-based Industrial Working Group initiative. For example, Airbus Industry is committed to using TOPCASED in its A350 program. At present, TOPCASED is evolving into OPEES (Open Platform for the Engineering of Embedded Systems), which is funded by the European Community’s ITEA research program (for more on OPEES see the following section of this document) as well as a consortium of industrial enterprises.

XMF is an Eclipse-based technology developed originally by the UK company Xactium, which eventually released it into open source through a non-commercial spin-off, Ceteva. It comprises a MOF-like language and corresponding facilities for the definition of the syntax and semantics of domain-specific modeling languages, and the automatic generation of corresponding run-time systems. Xactium had successfully applied XMF and related technologies to a number of industrial projects. However, it seems that the company could not achieve sufficient critical mass as a tool vendor, which led to a switch of its business focus from tools to services. Nevertheless, based on prior results, the technology itself looks quite promising for designing compact domain-specific languages.

Recommendations

All of the tools and projects identified above have been selected because they may prove interesting for one reason or another. A number of them provide capabilities that are complementary to other Eclipse-based modeling tools recommended in the previous section of this report. Consequently, we suggest a more in-depth investigation of each of these and, based on that, selecting a subset whose progress should be tracked further and, if appropriate, influenced. Of particular interest and relevance is the TOPCASED project with its related tools/projects and its follow-on OPEES in the real-time embedded Also, it may be useful to focus on one or more of the research projects, notably, Kermeta, Fujaba, and Coral, since they could be influenced to work on new capabilities.

Other Initiatives

This section considers other initiatives which aren't, in essence, a project or tool but which may have an important influence on the decision to adopt particular open source technologies.

OPEES

The Open Platform for Engineering of Embedded Systems (OPEES) is an initiative whose objective is to ensure the long-term availability of open source software technologies to secure industrial competitiveness and development. As an organization, it provides governance, a business model, and an ecosystem. As a technical repository, it provides a set of components, and incubation area, and a framework for integrating tool chains. OPEES also provides a set of methods and processes to ensure long term availability of its components and support the operations of the organization.

Recommendations
The mandate of OPEES is not dissimilar to that of Eclipse (or any ecosystem, for that matter), except that it is focused particularly on long term sustainability. One might question whether an initiative like would have more success if it adopted the governance model of another proven initiative rather than incurring the overhead of establishing its own; indeed, the initiative is in the process of exploring the possibility of forming an Eclipse Industry Working Group, but such a working group would have much overlap with another, similar initiative (see below), with which it should perhaps collaborate, if not merge.

Modeling Platform Working Group

The Eclipse Modeling Platform (EMP) is an industrial quality integrated software platform to enable a complete tool chain of model-centric tools used by organization focused on model-based engineering. The requirements for EMP are being set by some of the largest companies using model driven development. EMP will be developed as an Eclipse Industry Working Group supporting a collection of open source projects and encouraging a commercial ecosystem of value-added tools and services. The platform will be based primarily on existing Eclipse modeling technologies but focus on better integration, quality, scalability and usability for modeling professionals. Assuming the decision is made to proceed with formal creation of an Eclipse Industry Working Group (in June 2010), the goal is to release a first version of the platform by June of 2011.

Recommendations
In order to be successful, the Eclipse Modeling Platform must attract and retain a critical mass of major corporate adopters to support the platform through 1) direct investment and 2) procurement of value-added products and services from the commercial community. Initial participation in the pilot working group (by companies such as UBS, Ericsson, Alcatel-Lucent, SBB, SAP, SWIFT, Fraunhofer FOKUS, Euranova, and itemis) shows promise, but it remains to be seen whether these companies will be able to make the necessary long-term commitment. Nonetheless, even if all this working group were able to produce is a basic modeling IDE, that would still be quite a useful outcome.

Appendix A: Eclipse Project Capabil-ities

The accompanying Excel spreadsheet, `EclipseProjectCapabilities.xls`, provides a matrix indicating which Eclipse Modeling projects support which of the capabilities identified by the study (described earlier).

Projects shaded in grey were excluded from detailed evaluation because they are on the verge of termination/archival as per the Eclipse Development Process; similarly, projects that are not shaded (i.e., EMFT ECP, GMP Graphiti, and MDT Sphinx) were excluded because they are still in their Proposal phase. Those shaded in red were also excluded because they did not support any of the capabilities. Projects shaded in green were evaluated in more detail (see above and subsequent appendices) because they were identified as supporting one or more of the capabilities.

Appendix B: Eclipse Project Criteria

The accompanying Excel spreadsheet, `EclipseProjectCriteria.xls`, provides a matrix indicating high level "ratings" (see below) of selected Eclipse Modeling projects based on the criteria identified by the study (described earlier).

Ratings (3 for low, 6 for medium, 9 for high) were assigned to the projects based on how well they met the evaluation criteria, to provide a high-level indicating of the stability/quality of each project. Factors that contributed to the ratings included information gathered from readily available sources, input provided by project leads, inside knowledge of the motivations behind the companies involved in the projects, and experience using the frameworks/tools in developing proprietary tools. An overall rating was determined by calculating an (unweighted) average of the individual ratings for each project. An overall rating of 7 or more was considered high (and shaded in green); between 4 and 6, inclusive, was considered medium (and shaded in yellow); 3 or less was considered low (and shaded in red).

Projects shaded in grey were not rated because they were not selected for detailed evaluation (see above).

Appendix C: Eclipse Project Evaluations

The accompanying Word document, `EclipseProjectEvaluations.xls`, provides detailed information that was gathered and used to evaluated each of the projects selected by the study (because they supported one or more of the capabilities identified above).

The sources of information used in preparing this supporting document included (but was not limited to) the following:

Bugzilla
Likewise, Bugzilla (see https://bugs.eclipse.org/bugs/) was used as a source of metrics (provided by project leads) about the number of bugs against the various projects.

Dash Project
The Dash project at Eclipse (see http://dash.eclipse.org/) provided useful statistics on projects, including activity levels of committers and companies.

Developer Mailing Lists
In some cases, metrics were provided (by project leads) based on developer mailing list activity (see http://www.eclipse.org/mail/).

Download Statistics
Eclipse used to maintain statistics for downloads (see, e.g., http://wiki.eclipse.org/index.php/Project_Download_Stats), in various forms, although these are of questionable accuracy and may be less valuable than they used to be now that more people consume content via update sites, packages, and most recently, p2 repositories. Instead, information from Google Analytics was included for Modeling projects, where available. It is important to note that these statistics do not necessarily represent actual downloads, but, rather, the number of visitors to the project's download page.

Eclipsepedia
Eclipse projects are increasingly using the wiki as a place to put their up-to-date documentation. This was often a more valuable resource than the static home pages.

Newsgroups/Forums
Similarly, some project leads provided metrics based on the newgroups/forums (see http://www.eclipse.org/forums/) for their projects.

Ohloh
Ohloh (see http://www.ohloh.net/p) was used as a source of licensing and language information for projects. Ohloh searches the source code for individual license declarations, which can differ from the project's official license. It also determines the language of each line of code, excluding comments and blanks.

Project Leads
Project leads were asked to provide input on the various criteria (in particular, the technical criteria) for their projects.

Project Metadata
Each project is required to keep up to date information about their contributors and releases, referred to as "project metadata". This information was gathered from the Portal and the web. One interesting piece of information that was obtained from the portal is committer activity, i.e., how much have the various committers committed to a project over specific periods of time.

Project Plans
Project plans provided a sense for the roadmaps for the various projects, in the absence of more detailed insight on the web or wiki.

Project Web Pages
Information on the various projects is inconsistent at best, and in a lot of cases out of date, but some useful information was found on the project home pages.

Appendix D: Other Tool Capabilities

The accompanying Excel spreadsheet, `OtherToolCapabilities.xls`, provides a matrix indicating which of a number of tools outside of Eclipse support which of the capabilities identified by the study (described earlier).

Appendix E: Other Tool Criteria

The accompanying Excel spreadsheet, `OtherToolCriteria.xls`, provides a matrix indicating how well a number of tools outside of Eclipse support a subset of the criteria identified by the study (described earlier). The criteria used in this assessment are the same as those used for the Eclipse-based projects.