Eclipse Modeling Platform Requirements
Version 1.0
April 27, 2010

The following is a more detailed description of the requirements specified in the Modeling Platform (MP) vision white paper, dated Feb. 17.

Types of End Users of the Modeling Platform

The Eclipse Modeling Platform provides the foundation for a wide variety of integrated modeling capabilities. These capabilities are used by a variety of different categories of users. Understanding the idiosyncratic needs of each user category is critical in producing truly useful tools that do not suffer from the terminal complexity that characterizes many of today's commercial and open-source modeling tools. In this section, we briefly outline some of the most relevant categories of "end" users of the Modeling Platform. In the terminology of use cases, these categories represent different classes of actors. (Note that a given individual may play more than one of these roles.)

Tool designers are individuals who can create completely new tools or, more likely, add new capabilities to existing tools. This can be done in several ways, but invariably requires Eclipse programming skills and deeper knowledge of the platform its API, and even its implementation.

Language designers, on the other hand, typically use specialized domain-specific language (DSL) tools to (1) define or modify their languages and (2) generate suitable tools for those languages.

Tool adaptors are responsible for selecting and configuring one or more tools to be used for a particular project or team. This involves setting or selecting values for configurable tool options, defining the look and feel of the tool (where tools allow such customizations) to meet the needs of their user base. In general, this does not require deep knowledge of the tool implementation or sophisticated programming skills, since these individuals work through the user interfaces provided by the tools.

Artifact authors use Modeling Platform tools to create and modify the artifacts. "Artifacts" includes models, of course, but are not restricted to just those. The artifacts generated depend on the purpose of the authoring tools. For example, some tools are used to create document artifacts; others model transforms, code generators, etc. Artifact authors are domain experts that generally have some expertise in using the authoring tools and adapting them to their individual needs (within the confines set by Tool Adaptors). However, they should not be expected to have any understanding of the implementations of the tools they are using or the Modeling Platform in general. In some cases, such as system designers, they may not even have any programming skills. A more detailed breakdown of this category will be required when individual tool use cases are defined.

Artifact readers/inspectors do not create artifacts, but do use tools to inspect and/or analyze artifacts produced by corresponding artifact authors. They are domain experts but not necessarily tool experts. Consequently, they need relatively unsophisticated tool interfaces that allow them easy access to the information they are seeking and in the form that suits their purposes. As in the case of Artifact Authors, this category is tool dependent and will need to be refined, as more detailed use cases are evolved.

Requirements

Model Version Management (Life-cycle Support)

1. Managing versions on various model granularity, meta-model and instances

1.1 The MP needs to be able to support versioning of meta-models and instances. The important aspect is that all types of users can specify the granularity of version and what objects are to be versioned.

1.2 The versioning features needs to supporting multi-user and distributed development teams. It should allow different users or teams to create branches of model instances.

1.3 Support for different types of model repositories should be possible.

Use Cases

UBS:
UBS plans a model-centric approach to maintain the concept of the enterprise architecture. The overall meta-model of the enterprise is a collection of many small meta-models. A small meta-model can have relations to other small meta-models. The MP needs to support version of meta-models (single meta-model aswell as a set of related and depended meta-models) and the instances based on these meta-models. An instance can be a single object or a set of related object. Meta-models in the terminology of UBS are domain specific language (DSL).
→ See Flexible Content Support → Domain Specific Language

Use Case: Edit model concurrently
Actors: *Artifact author*
Two artifact authors edit concurrently the same model instance. If they work on different objects, there should be no conflict. If they edit the same objects, the platform can identify the conflicting changes and merge them interactively with the user to resolve conflict.

Use Case: Configure versioning support
Actors: *Tool Adaptors*
The tool adapter, in cooperation with the language designer, decides on the level at which objects must be versioned. Objects that are not in classes with versioning support will not have a full history: every change to them will cause the object to be removed and recreated.

Use case: Evaluate changes between published versions of models
Actors: *Artifact readers/inspector*

The artifact author performs a series of changes to an existing model instance. After quality review, those changes are made available to external parties (artifact reader), with enough decoration to determine all changes that have been performed on the model instance, on an object level. The artifact reader typically wants to (i) isolate those changes, (ii) review them, (iii) understand the rational behind every change and the impact on other model elements (across metamodels, typically) in order to (iv) determine the relevance of these changes on its particular context (implementation, etc).

The information required to perform this kind of analysis can contain:

- a collection of all the changes performed (like a transaction log);
- a list of changed objects per transaction, with a change description;
- per change, a "change classification" that describes the kind of change that has been applied (e.g. adding a code to a code list, changing a service definition in a backward compatible way).
- the reason of the change (business rationale, eg as retrieved from a change request management system).

2. Analysis of model changes

2.1 The MP should provide the ability to track and report changes between two different versions of a model. A change to a model could trigger a file output that describes the change. User should be able to predefine changes patterns for a model and have the MP generate triggers when those patterns change.

Use Cases

ALU :
Actors: tool designers, artifact authors

Artifact authors use a model driven IDE to create several kinds of JAIN SLEE components and put them together to build applications.
A meta-model has been defined for each type of this artifact. The application development process is the following :

1) Create new components (models)
2) Generate Java sources (based on JET templates)
3) Build binaries
4) Package the application

When the definition of a component changes (impact on meta-model or model), the existing sources & binaries of this component and the application package that includes it are no longer in line with its new definition
The tool designers should provide the capability in the IDE of detecting these incoherencies and triggering (on demand) automatic processes to regenerate sources (when required), rebuild the binaries (when required) and repackage the application. Artifact authors can trigger the analysis of the model changes and the upgrade process when they have changed

the components.
In order to understand what it needs to do, the IDE should base its analysis on some input files that describe the changes performed on the meta-model/model of the component(s).

3. Merging of different models and version of models

3.1 Different teams and individuals should be able to work on different parts of a model and then be able to merge the changes together into a common model. Part of the merge will include consistency detection on the merge model and flags will identify inconsistencies.

3.2 Model merge must support the merging of concrete syntax (graphical or textual) and abstract syntax (semantics). For concrete syntax, persistence of representation (ie, diagram layout) is important during the merge.

Use Case

ALU:
Actors: tool designers, artifact authors

Tool designers created a script editor (workflow tool) by using EMF & GEF.
The script editor is defined by a meta-model and each script logic created with this tool is an instance of this meta-model.

The application logic is the script logic and is triggered by events.
In the script itself, a connector initiates the logic for each event that can be handled by the application.
Several artifact authors can work on this script, each of them defining the logic for a specific event.
When the developments of the different logics are completed, the artifact authors execute the merge of the different script versions into a single one by using the tool.
Tool designers have to provide an automatic merge within the script editor.

4 Change analysis to identify root cause and impact analysis for changes to all levels of model content.

4.1 Artifact author/readers should be able to see the impact of a model change might have of other models, implicit or explicit. The impact analysis should support changes at any level of a model.

4.2 Artifact author/readers should also be able to link a change request that was implemented for a specific release to the specific changes that were the results of this request.

4.3 The change request will be stored in an external change request system. Therefore, support for different change request systems will be required.

Use Case

UBS: As described in Chapter 1, UBS plans to have an overall meta-model of the enterprise as a collection of many small meta-models. The overall meta-model contains concept defined in the software development lifecycle (e.g requirement model, analysis model, design model, test model, deployment model, reference architecture model). A concept can be connected to other concept. A change of a concept can impact other concepts. (e.g if a requirement gets extended, this can impact the analysis model, design model, test model …). It would be great if we can capture such impact. This leads to the requirement of connect object in a model to other objects → Chapter 5

5. Traceability to identify connection between model elements.

5.1 The MP must support the ability to create relationships between models and model artifacts that are independent of the meta-model.   Tools must be able to show these relationships and provide a level a traceability between the model elements.

5.2 Artifact authors should be able to specify the relationships a high level and then elaborate in more detail and provide traceability a different levels of the model hierarchy.

5.3 The MP must support the traceability between non-modelled artifacts and various types of models and elements/parts of models.

Use Case

Ericsson
For example, requirements might not be modelled but we still need to be able to trace from them to various types of models and elements/parts of models.

An example can be to trace from a requirement to models/model elements where the requirement is designed and implemented and to models where it is verified.
And vice versa from design, implementation and verification models to see the requirements from which they originate.

In the most simple case, it would be sufficient to be able to follow the trace, e.g. open the requirement from the modelling tool or the model from the requirement tool.
What could also be desirable is the any change of an artifact is reflected in the connected artifact, e.g. using a listener/notifier functionality.

6. Support for meta-model change and update to appropriate instances.

The MP must support the ability to change a meta model and then apply the change to the model instances. It should show what changes are possible and which ones are not.

The MP must support the ability to have two different version of a meta model in a single environment. The artifact author should be able selectively migrate an model instance to a new version of the meta model.

Use Case

ALU:
Actors: tool designers, artifact authors

Tool designers created a script editor (workflow tool) by using EMF & GEF
The script editor is defined by a meta-model and each script logic created with this tool is an instance of this meta-model.

In version 1 of the script editor, several scripts are created by artifact authors.
Then, the version 2 of the tool comes with new features that impact its meta-model.

Backward compatibility needs to be guaranteed.
All the scripts created with Script Editor v1 must still be editable in the Script Editor v2.
For this purpose, the Script Editor v2 should detect that the script (model instance) has been created in a previous version and should migrate it to the new version of the tool (new version of the meta-model). Artifact authors should not be involved at all in this migration.
Tool designers need to provide an automatic upgrade process.

UBS:

- Record history of evolution of a meta-model as a change model which allows applying the changes to model instances automatically or with manual interception if automatically is not possible. (Depends on the technical serialization)
- Being able to evaluate the difference between two version of a (meta-)model.

Model Audit Support

1. Overall end-to-end life cycle management / model governance (elements can be in different 'review' states)

1.1 The MP must support the concept of review cycles and approvals.   It should be possible to record who reviewed, approved and comments for each cycle.   The MP should also support the ability to specifiy who needs to review and who can see a model.

1.2 A side effect of this requirement is that the MP will need to concept of a user and role. A system of authentication and user management will also be required.

[NOTE: We should investigate the use of external authentication systems]

2. Quality Checks for Models

2.1 The MP should also allow for a reviewer to provide a quality check during the life cycle of a model. The quality check should be saved as an assurance record with the model.

Use Case

UBS:
Review of model parts is orthogonal to the model content:

- Pluggable review to specific parts of a model
- Review (Request, Approve, Dissapprove)
- Freezes reviewed part of a model

Use Case: Promotion/Review of content and limited visibility
Actors: *Artifact author, Artifact reader*
The artifact author modifies an object. This object, in its new version, is not immediately visible to everyone. It must first undergo review and formal promotion to different visibility levels (like integration of development streams, for example). The artifact reader (for example, a member of the author's development group) reviews it, deems it to be OK and the object gets visible to a larger audience (typically, the full organization or even external artifact readers).
The review result, promotion and decision by the artifact reader is linked to the object, for auditing reasons. This is to ensure that the process that has been followed complies with the model development standards.

Core Platform Features for Enterprise Use

1. Support for large models, including lazy loading, partial collection loading

1.1 It is expected the MP will need to support models contains 500K plus elements. A key aspect is that the tools built on the MP will scale to support large models. For instances, a diagram editor will continue to function efficiently on very large models.

Use Case

SWIFT

- Current # EObjects in our models ~ 80 000 (in 2 resources)
- New content to be captured (~ 300 000 objects) (can be in lots of resources : ~7000).
- Expected updated objects every year: 15 %
- Expected created objects every year: 10%

So, around 500 000 objects within 5 years, with about 60 000 changes to be recorded by the versioning system.

Ericsson
- For some of our system models we have, the number of diagram are in the region of 2000-3000 sequence diagrams
- For ordinary UML/class models I would estimate a model in Eclipse would weigh in around 150-450mb on disc depending on size and contain 2000-10000 elements (classes, packages and protocols). Depending on the granularity of elements you count it could be much more for sure.

UBS
- Current # EObjects in our Model ~ 200 000 and # EReferences ~ 900 000 (for Domain COBOL Reverse Engineering)

2. Ability to work offline and then merge changes back into a model

2.1 An artifact author/reviewer should be able to extract a partial view of a model repository that could be used by an external editor. The artifact author/reviewer should also be able to import and updated view back into the model repository.

2.2 All types of users should be able to continue to use their MP tools without be connected to the network. A partial view of the model repository should be able to be used off line with the MP tools.

Use Cases

Use Case: Scoping of content
Actors: Artifact Author
The author defines a view that only contains a "scoped" version of a model. For example, only the messages/service specification that make sense in the scope of a project. The system extracts this view to a standalone consistent resource and makes all changes into that scoped resource. This can happen in several different tools, possibly outside an Eclipse IDE, provided the editor is capable of updating the resource content. The platform reads the resource content and updates the repository content with it, possibly via a "merge" mechanism that only records the delta/changes effectively implemented.
This "scoping" mechanism makes it possible to keep the overall model to a reasonable size, both for performance and readability reasons, without having to physically partition a model into resources. Defining a "scope" should be similar to content tagging.

3. Support for modeling standards and open formats. Lowest denominator is the EObject (ECore)

3.1 The MP should assume the lowest common denominator is Eobject in Ecore. Tool adopters should be able to build/add support for model standards and formats at any time. The tools provide the MP should continue to support these new model definitions.

4. End-to-End project support from business architecture to code and testing

4.1 The MP must support for different types of models during the different steps of the software development lifecycle, ex requirements, Use Cases, Testing, and Code.   Tools such as traceability and impact analysts should work across the different types of models.

Flexible Content Support

1. General purpose models based on industry standards like UML, BPMN and SYSML

1.1 The MP must support out of the box general purpose models based on industry standards like UML, BPMN and SYSML. Tool adopters should be able to extend the support to any modeling standard.

2. Domain Specific Models

2.1 Language designers should be able to define a modeling language for a specific domain. The MP features should work for the new domain specific model.

2.2 Domain specific language clearly defines rules for the syntax (grammar) and the semantic (sense). The syntax allows determining if a model instance is syntactical right without any error. The semantic rules define the interpretation of the formal model to extract the model specific information. This extraction is unique and reproducible. The MP provides the following feature:

- Define DSLs (meta-models)
- Maintain instances of DSLs
- Validate instances against DSLs

3. M2M Transformations

Model to model transformations are useful in a variety of situations. In particular, they are useful when a model in one language needs to be transformed to a model in a different language, although they can also be used to generate subset models (e.g., views) of the originals, based on some selection criteria (e.g., a viewpoint). The most common use case for model transformations is where a design model is transformed into an analysis model for a specific type of analysis, such as, for example, the transformation of a UML model into a queueing network model, which can then be used to predict the performance characteristics of the design.

3.1 **Use case**: Create or update a model transformation
Actor: *Model transform designer*

The model transform designer must be able to create a new model transform (usually based on the source and target metamodels as inputs) or to update existing ones and to test them for validity.

3.2 **Use case:** Perform a model transformation on one or more models
Actor: *Model analyst* (i.e., this could be a model author, or some analysis method specialist (e.g., performance analysis expert), a (model) tester, or even a model reviewer)

The model analyst should be able to select and invoke an appropriate model transformation on one or more models and obtain access to the transformed model as well as to a summary of the results of the transformation (e.g., error logs). This may require prior annotation of the original model, which requires suitable annotation capabilities in the model authoring tools. In addition, for those transformations that require human intervention, the model analyst should be able to interact with the transformation engine and guide it as necessary.

4. Model to Text (M2T) Transformations

4.1 It should be possible to generate source code, documentation and/or test cases from a model.

Governance

This refers to two distinct capabilities:

(1) tracking and management of project status
(2) operations, administration, and management of a modeling tools suite used by a project or enterprise

1. Tracking and Managing Project Status
This requires a project status "console" where the current status of a project can be monitored and project history tracked.

1.1 **Use case:** Define which project information is to be tracked (e.g., number of error reports, loadbuild statistics, etc.) and how it is to be reported
Actor: *Project manager*

The project manager must be able to define which information is to be collected and when and also how it is to be reported (e.g., definition of metrics). This implies the ability to collect such information from various tools within the suite.

1.2 **Use case:** Monitor project status
Actor: *Project manager*

The project manager should be able to access a project console (customized for specific project needs) and be able to determine the current status of key project indicators as well as how they evolved over time (project history). Furthermore, it should be possible for the tool suite to raise warnings when certain critical situations or events occur.

2. Toolset Configuration and Management

This functionality involves the static configuration and dynamic management (e.g., provisioning, adding/removing tools and tool versions, configuration) of a tools suite consisting of tools that are based on or part of the Modeling Platform. It provides a common decision and control point for consistent management and customization of a tools suite that is shared by a project team or an enterprise. The assumption here is that individual projects and enterprises will often need to assemble their own configuration of tools and customize them and operate them in a manner that suits their specific needs. Over time, the configuration of such a tools suite will need to be updated (e.g., providing new versions of existing tools, adding new tool capabilities, etc.) in a controlled fashion, so that all users of the tools suite can be synchronized.

2.1 **Use case:** Configure a tools suite for use by a team or enterprise
Actor: *Tool adaptor*

The tool adaptor selects the set of tools required in the tools suite and configures their parameters and other customization settings to suit the needs of the project or enterprise.

2.2 **Use case:** Add a new tool/capability to the tools suite
Actors: *Tools suite operator, Tool adaptor*

The tool adaptor selects a new tool/capability and configures it for use within the tools suite. The tool operator integrates the new tool into the current suite and activates it.

2.3 **Use case:** Change the version of an existing tool in the suite
Actors: *Tools suite operator, Tool adaptor*

The tool adaptor selects the new version of a tool and configures it for use within the tools suite. The tool operator integrates the new tool version into the current suite and activates it (this may involve invoking automated model updates).

=== End of Document ===