Impact Control

Open Integration Platform

Meta Model Based Tool Coupling

Cross-Tool Traceability and Baselining

And More
Open Integration Platform Impact Control

**Requirements for Mapping Processes**

Whether the processes employed are more agile or more formal, every software developing organization must provide for proper methods, good tool-support and an effective roll out strategy.

The aim is to provide continuous tool support for all phases of the development process, including all of requirements management, the design phase, implementation planning, the actual production of software and finally testing of the initially defined requirements.

- **openness and vendor independence**
- **incremental and flexible expandability with high levels of integration**
- **traceability and use of baselining across the entire process**
- **based on cutting edge open source tools**
- **satisfaction of accepted standards for processes and technology**

**Basic Approach: Requirements Driven**

With a requirements-driven approach alone, many of above criteria can be met. Requirements should transparently "flow" through the entire development process, all the way from their emergence to their release (whether they are new requirements, change requests or reported errors).

It should be perfectly clear at all times:

- Where a requirement originated
- Which state of progress the requirement is in, and who has been working on it
- How a requirement is implemented and in which version it features
- Whether a requirement complements or is derived from others

Irrespective of the choice of process model, a development process diagram will resemble fig. 1:

![Development Process Diagram](image-url)

**Figure 1:** building blocks of software development, process view
High Level Use Cases According to Impact Control

Project Structure Creation
- Design of the project structure (process and development structures) -> definable templates

Requirements Management
- Import/export of requirements
- Synchronize requirements with an external tool (bidirectionally)
- As necessary, create and edit requirements via an internal editor

Design and Modeling
- Synchronize UML diagrams with an external tool (bidirectionally)
- As necessary, create and edit UML diagrams via an Eclipse-based internal editor

Automation of Modeling and Development Steps
- Using QVT (Query View Transformation), automatically generate modeling and source code artefacts according to predefinable transformation rules, e.g. generate a use case for each requirement or vice versa

Consistency Checks
- Define and verify consistency rules using OCL (Object Constraint Language), e.g. verify that each requirement has at least one use case

Traceability Management
- Create traceability links between artefacts
- Visualize links between random artefacts using Matrix
- Correct and/or create new traceability links directly in Matrix

Change Management
- Synchronize change requests with an external issue tracker
- Link change requests to edited artefacts using a task based commit

Impact Analysis
- Display direct dependencies
- Display indirect connections (up to the process' beginning/end)

Baseline and Release Management
- Create baselines
- Relate artefacts to baselines
- Assign states to baselines, e.g. “planned”, “test”, “invalid” or “released”
- Display existing baselines and their current state
- Display the artefacts for a given baseline
- Compare two baselines by the change requests to its artefacts (including models) and their relations to another
- Check out baseline-defined configurations from the repository.
- Compose a production baseline from the baselines of single components (Hierarchical Baselines)
Version Control
- Persist all relevant artefacts (configuration items) in a central repository

Software Development
- Task-oriented
- Analyze source code with respect to Cyclomatic Complexity
- Reverse engineering

Parallel Development of Artefacts including Models
- Implement competing changes on the same line of development
- Compare models (visualize differences)
- Merge models (compose models selectively)

Build and Test Management
- Use integrated open source and/or commercial tools to build and perform tests

Project Management
- Synchronize tasks with an external project management tool for iteration and release planning

Document Generation
- Generate information on artefacts from various process phases into predefined templates, in order to exchange requirements or design documents with third parties

Code and Configuration Generation
- Generate applications on the basis of both modeling from earlier process phases and input about configurations and variants
Meta Model Based Tool Coupling

Basic Concept and Overview

Typically, software development processes involve a large number of players, who contribute to software production on various abstraction levels and using various tools. As each tool is instructed by a supervisor to process information from upstream sub-processes, it produces output information, or results, for downstream activities. In practice, natural incompatibilities will arise, e.g. for the following reasons:

- Tools have limitations on the exchange of information.
- Parts of the implementation activities are not traceable because information is being exchanged manually.
- Tools manage their output information in isolated private repositories (such as databases, configuration files, internal file formats etc.), across various operating systems. The relations to output information of other tools cannot be mapped (lack of cross-tool traceability).

In order to overcome above problems, a meta model based tool integration system is favoured. The key element here is the mapping of tool-specific information onto a unified meta model, which provides all of the language constructs required for a presentation of the entire development structure with all its processes.

![Figure 3: principle of tool coupling by meta model](image-url)
The various bits of information output by respective tools (model instances) as well as the relations (links) between them are stored persistently in the repository. It is thus vital that tools are able to exchange information via import/export interfaces.

Within the context of software development, a good choice for meta model representation is UML, which uses well established concepts for the description of software systems. On this basis, the meta model is expanded by introducing further constructs (requirements, issues, test cases, etc.) that represent the outputs from each respective phase of development.

 Depending on the required level of detail, elements are modeled either finely grained or as larger blocks. The relations between model blocks are represented by UML associations/links or own specific types.

Fig. 4 gives an example of some tools that are typically used for the respective phases of development.

Within the framework of requirements analysis, functional and non-functional requirements are created that specify the properties of the system that is to be implemented. In the next phase, these requirements are refined with use cases and activity diagrams. A coherent process requires the traceability of these phases, meaning that it must be possible to review which requirement initiated which use case (and vice versa) at any later stage.
With a meta model based approach, this information can be managed securely and persistently, especially because the relations are stored as model instances with the output results. In addition, the creation of use cases from requirements can be automated using model transformations.

Recognized meta models such as UML or CWM are based on the MOF (Meta Object Facility) meta model, developed by the Object Management Group (OMG).

The OMG consortium was founded in 1989, and deals with the development of standards for vendor independent, cross-operating system and object oriented programming. The OMG was founded in support of 11 brands including IBM, Apple and Sun. It now has over 800 members and the standards it develops are internationally recognized.

The Meta Object Facilities feature further specifications such as QVT (model transformation) and OCL (invariants and conditions). With these technologies, model elements can be automatically generated from other elements by transformation and tested for model consistency.

As an example from requirements analysis, a requirement is created in Word and is imported to the repository via the OpenXML interface. By model transformation, each requirement is generated into a use case (with the same name, for the sake of argument). The generated use cases are then further refined by using a UML tool via the XMI exchange-format. The invariant “each requirement is refined into a use case” can be verified at any time by a model inquiry. In this way, model inconsistencies can be easily identified and corrected at an early stage.

This concept can be consistently applied to all other development phases. With each development step, the repository is further expanded to include new model elements and relations.

The repository is managed by an impact control center (ICC). This ICC puts in place and uses the interfaces to external tools and provides the functions for carrying out model transformations and queries.
Meta model: Software Development Process

A meta model is conceptualized for each individual development phase describing typical artefacts, outputs and relations of the activities involved.

The conceptional meta model in fig. 6 serves as an example of traceability of the relations between the phases of requirements, design, construction and testing.

![Meta model diagram](image)

Figure 6: representation of traceability in a meta model

By modeling relations between the distinct phases, “rules” are established which must be followed by the entire development process, ensuring consistency:

- Each requirement is refined by a use case
- The behaviour of a use case scenario is described by activities (an activity diagram)
- Activities are represented by the operations of a component
- Each design component is realized in its concrete implementation
- Requirements and components are verified in an appropriate test case

For example, a modeled component which has not been verified in a test case must not be released for implementation.

Option: Model Driven Engineering

The repository contains a platform independent model (PIM) of the software to be developed. The system may be specified by a UML component model. The platform-specific model (PSM) may be realized on an application server using Java Enterprise Beans. Via QVT, large sections of the Java code can be automatically generated from the component model using standard transformations.
Technical Realization: Eclipse

Eclipse is now one of the biggest open source projects worldwide. It integrates a great number of frameworks, covering almost every area of software development.

This rich environment currently provides all of the technologies needed for the concrete realization of an integration platform and a model based repository:

- Implementation of the MOF specifications using the Eclipse modeling framework (Ecore, EMF, GMF, MDT, QVT, OCL)
- Application development with Rich Client Platforms (RCP, SWT)
- Java-IDE for developing tool adapters
- Expandability with Eclipse plug-ins
- UML 2.1 plug-in for design model composition

The impact control center is realized as an RCP application that can be integrated with the Eclipse IDE by using a plug-in. The graphical user interface (Standard Widget Toolkit) visualizes the model elements of the repository and provides various navigational views for the entire model. The model elements are based on Ecore (Eclipse MOF) and are seamlessly integrated using the EMF framework.

Figure 7: screenshot: views
The repository is navigated by a tree view that structures logical sub-models and packages according to the project's requirements.

![Repository Navigation](image1)

Figure 8: screenshot: repository navigation

The model elements' dependencies can be traced at all times, e.g. by specialized views of the model.

![Relationships](image2)

Figure 9: screenshot: relationships (e.g. requirement -> use case)

The Eclipse environment delivers highly sophisticated frameworks for model validation (OCL) and transformation (QVT). Via integrated menus model inconsistencies can be revealed, visualized and traced to distinct process phases at any given time.

Versioning of models is facilitated by an appropriate team plug-in (Subclipse, Subversive, etc.).
Future developments by the Eclipse community in modeling and MDA can be integrated with a low level of adjustment effort and may enhance the software development process with new and practical features.

![Figure 10: screenshot: tool integration (e.g. issue tracker)](image)

Integration of the various tools is implemented using adapters. For example, web based issue trackers such as Trac or Bugzilla offer an XMLRpc interface. Using such interfaces, adapters convert between tool-specific data formats and an equivalent Ecore format. Introduction of a new tool simply requires the development of a new adapter, while the core of the model remains unchanged.

**Conclusion**

In summary, it can be concluded that, due to Ecore standardization and the open possibilities for expansion, implementing the concept of meta model based repositories in Eclipse provides a real alternative to monolithic “all-in-one” tools.