

RAD Technologies

---

# MDD Enablement Tooling

*Version: February 4, 2004*

Authors: Yael Shaham-Gafny  
Shiri Kremer-Davidson

Contact: Yael Shaham-Gafny (Yael.Shaham-Gafny@haifa.ibm.com)

**OVERVIEW .....**

**TOOLING TOPICS .....**

Model browsing and navigation .....

Consistency checking.....

Anti-patterns.....

Model validation.....

Conventional (code) debugging with UML notation.....

Metrics.....

Understanding model changes.....

**THE STATE OF THE ART .....**

In the Industry .....

In the Research Community .....

**APPENDIX A: STATE OF THE ART IN INDUSTRY .....**

AllFusion Component Modeler .....

ArcStyler.....

iUML (Executable UML) iUML Modeler & iUML Simulator .....

NetBeans Metadata Repository (MDR) Project.....

Objectteering UML.....

Poseidon .....

Rhapsody.....

SDMetrics.....

Tau Generation 2.....

Together .....

WayPointer .....

XDE .....

**REFERENCES.....**

---

# MDD Enablement Tooling

## Overview

Model Driven Development postulates that the main artifact being developed is the model rather than the code. All the advanced technologies that support productive code development, such as interactive visual debugging, static analysis, code navigation, etc. will be needed in the context of model development. In this document we survey the state of the art in the area of MDD enablement tooling, in both industry and academy. We present our vision on what is needed in this area to support the MDD strategy. In the first section we present our vision of the various topics that we see as important and challenging and in the second section we give a brief overview of the state of the art.

## Tooling Topics

### Model browsing and navigation

One of the important features of modern IDEs is their support for understanding code. The developer can browse the code, follow dependencies, perform queries (textual and structural), etc. We need the same kind of support for understanding models. This includes capabilities such as:

- **Interactive model browsing:** Allow users to create views (diagrams) on the model interactively according to various criteria. Some criteria can be predefined (such as slices), and in other cases the user can define the criteria, for example by selecting elements and what relationships to view. The diagrams are constructed on the fly according to the current browsing needs.
- **Grouping:** A group is a set of model elements selected by the user. The user can then view relationships between groups, or view relationships between an element and a group. For example visualize relationships between controller parts and view parts (in an MVC implementation).
- **Zooming:** drill down into composite state, interaction fragment, etc.
- **Following “references”:** This mechanism enables navigation from the definition of a model element to places where it is used and vice versa. For example: between a message in an interaction and the class method; between a class attribute and its uses in actions/guards etc.
- **Search & queries:** Similar to the Eclipse search feature, the user enters search criteria and gets a list of occurrences. In addition to the usual textual search, model searching can include queries on model structure and according to the meta-model semantics. Some examples: find all the references to elements of package x in elements of package y; find all singleton stereotyped classes.

Some of these capabilities are already present in existing tools, some require mostly a development effort, but some (such as interactive browsing and grouping) require additional research.

OMG MOF2.0 QVT (Query/View/Transformation) is considering proposals that provide query, view, and transformation capabilities which are pertained to be the main issues in the manipulation of MOF models. Some of the items above are coincident with the issues

QVT is concerned with, and can possibly share similar technologies. Our focus is on model visualization and tooling whereas OMG's focus is more on model manipulation.

## Consistency checking

The consistency problem we refer to here has been termed the *horizontal consistency* problem in the literature (more on consistency problems can be found in section "In the Research Community"). It arises when parts of a system are modeled from different viewpoints and there is a certain overlap between the viewpoints. For example in UML, sequence diagrams and collaboration diagrams are both used for describing object interactions, from a temporal viewpoint and a structural viewpoint respectively. As a consequence there may be consistency problems between these two behavioral models.

The major task here is to define consistency rules for the various artifacts of UML 2.0:

- StateMachine vs. Interaction
- Static Model vs. Object diagram
- Static Model vs. Interaction
- Static Model vs. StateMachine
- Interaction vs. Interaction
- Composite structure vs. Classes
- Etc.

The preferred approach for implementation would be to extend the existing capabilities of the Aurora EMF model validation. Consistency rules should be expressed in OCL when possible or in Java when OCL is not expressive enough.

Note: In some cases consistency and completeness are overlapping concepts.

## Anti-patterns

An anti-pattern is a pattern that describes a bad solution for a problem, explains why the bad solution looks attractive, why it turns out to be bad, and what alternative patterns are applicable instead.

The use of anti-patterns in design is not yet widely adopted by the software development community (as are patterns), partially because of the lack of supporting tools. Identifying an anti-pattern in the design requires expertise, and if organizations had this expertise they would not have applied the bad solutions in the first place. This is where tooling can help, by identifying problems in the early stages of the design, suggesting solutions, and thus elevating the design quality, educating developers and raising their level of expertise.

We see the following challenges in providing tool support for anti-patterns:

- Matching: finding occurrences of anti-patterns in the model.
- Representation: defining appropriate representations for anti-patterns that facilitate the matching. We need to investigate whether the existing representations for patterns are suitable in this regard.
- Refactoring: representing refactoring rules and applying them.
- Gathering a substantial set of anti-patterns.

The solution for anti-patterns needs to support extensibility, so that adding new anti-patterns in the future would be a simple task.

## **Model validation**

The term validation, when used in the context of code, usually means testing the implementation against the requirements specification for conformance. In the context of a UML model we see two aspects:

1) Testing the static portions of a model

The goal is to check the class model in order to see that it adequately describes the world that is being modeled. One approach is to semi automatically create object diagrams of two kinds. The first kind describes situations that are expected, and the second describes invalid situations. These object diagrams are then checked for consistency with the static model (Class structure, OCL, etc.). This method can help find over and under constraining of the static model. Furthermore, as a next step, an object diagram can be checked against the behavioral portion to analyze if the application can evolve to the situation described by it.

2) Testing the behavioral portions of the model

In order to test the behavioral portions of the model we would like to execute them. Up until now, model execution has been equated with executing the state chart portions of the model. The approach of developing a system by defining its behaviour through state charts is well fitting for real time applications. There is much controversy in the MDD community whether this methodology is applicable for all software development. The problem of finding suitable paradigms for model testing and execution for web and business applications is still open. The issues here are defining execution semantics for Activities, Scenarios, and simulating action semantics. The object diagrams described in the previous item can serve as an initial state (test setup) for testing the behavior.

## **Conventional (code) debugging with UML notation**

The idea is to bring the notions of abstraction and visual representation to the level of code debugging. This has a double advantage:

- 1) Helps the developer focus on the problem and masks low level, code based detail
- 2) Brings the developers into the world of models through concepts they are already familiar with.

IBM Rational XDE already has some support for this through a feature called Visual Trace. This feature allows the developer to view the debug trace as a sequence diagram. We propose to widen the scope of model driven debugging with the following features:

- Inspection: show a snapshot of instances and their relations using object diagrams. Such snapshots can be focused and filtered (by the user) to show relevant information.
- State: for objects that have an associated State Machine, show the current state.
- Scenario identification: compare user defined sequence diagrams with Visual Trace generated sequence diagrams to identify the scenario context.

## **Metrics**

Metrics measure the quality of a model according to specific criteria. Most criteria are applied to the structure of the model. Metrics are used to measure complexity, cohesion,

stability, testability, etc. There exists a variety of software metrics that characterize properties of a software system or an object-oriented design. Most of these are implementation level metrics that are computed from code. Some tools in the market already support this level of metrics for UML models.

The first stage of adding metrics support to IBM Rational's toolset would be applying general object-oriented metrics to UML. To achieve this, a technology for model analysis is needed. In addition the subject of visualization of the metrics information needs to be addressed in the context of models rather than code. Understanding the implications of the metrics will require elaborate model browsing and navigation such as we have described above. In the second stage it is important to identify metrics that are specific to the design level.

## Understanding model changes

IDEs coupled with configuration management tools provide excellent support for understanding code changes. We need the same capabilities for models, from two points of view:

- 1) Identifying differences between versions of the same model (To what extent is Aurora is handling this? We have ongoing discussions with the Aurora team with this regard).
- 2) Understanding the evolution of a model through the different development stages: analysis, design, implementation

The main challenge here is providing a meaningful analysis and visualization that enhances the ease of comprehending the differences. Such visualization would need to address the aspect of model changes and the aspect of diagram changes as well.

Another challenge that is relevant for the second point of view is extracting the model transformations that are manifested by changes.

## The state of the art

### In the Industry

We investigated how commercial tools are doing in each of the topics we listed above. A more detailed description can be found in the appendix:

#### **Model Browsing and Navigation**

No tool provides support such as we have envisioned. Some tools provide a search facility, few tools provide navigation via model relationships, and one tool (Rhapsody) provides automatic diagram layout.

#### **Metrics and Anti-patterns**

Most tools do not provide support for metrics. Some tools provide the conventional code and object oriented metrics tailored for models. SDMetrics is the only tool we have found to provide UML specific metrics. There is no tool that formally supports anti patterns though we found one tool (Together) that provides audit capabilities that can be considered in the same realm.

#### **Model Validation**

Most tools provide syntactic correctness checking according to the UML well formedness rules, and provide warnings for completeness problems. The popular general purpose modelling tools do not support any further form of model analysis/validation.

Model execution in the form of executing the state chart portions of the model is available in the context of tools for the realtime & embedded market, for example: Rose RT, Telelogic Tau Generation 2, Rhapsody from iLogix, and ModelRun by BoldSoft.

## Understanding Model Changes

Most tools support the basic team capabilities (versioning, checkin etc.), some tools support comparing versions and visualizing the differences, few tools support merging.

The following table summarizes the tools that we looked at. A more detailed assesment can be found in Appendix A.

Tool	Model Validation	Metrics	Anti-Patterns	Model Browsing & Navigation	Model Difference Visualization
AllFusion Component Modeler	✓ <sup>1</sup>				
ArcStyler	✓ <sup>1</sup>				
iUML	✓				✓
NetBeans	✓ <sup>1</sup>	N/A	N/A	N/A	N/A
Objectteering UML	✓ <sup>1</sup>	✓			
Poseidon	✓ <sup>1</sup>			✓ <sup>2</sup>	✓
Rhapsody	✓			✓	
SD Metrics		✓			✓
Tau Generation 2	✓			✓	✓
Together	✓ <sup>1</sup>	✓	✓ <sup>2</sup>	✓	
WayPointer	✓ <sup>1</sup>			N/A	N/A
XDE	✓ <sup>1</sup>				

<sup>1</sup> Supports only checking of well formedness rules

<sup>2</sup> Partial support

## In the Research Community

### Model Validation

There is much work going on in the research community on formal verification of UML models. In general this involves translating a UML model into some form or language that can be fed to a model checker or a theorem prover, and then running automatic verification to prove that certain semantic correctness properties hold for the model. The common properties that this type of verification can check for are:

- Liveness properties that describe something that will happen.
- Safety properties which are invariants that must hold in each state of the system.
- Deadlock which is a situation in which no transitions are enabled at the current state.

The drawback of this approach is that it is subject to the state space explosion problem, and can typically work only for small examples. Another drawback is that it focuses mostly on verifying UML statecharts, though some papers show how other artifacts of UML such as class diagrams can also be checked.

Another area that researchers have been active in is the ability to check a UML model with regard to a set of rules provided in OCL. This method can be used on the UML metamodel, and thus provide means to perform well formedness checking of a UML model.

### Consistency

The [Consistency UML group](#) gathers researchers from both industry and academia, who are interested in consistency problems in UML-based software development. A number of different consistency notions have been identified:

The *horizontal consistency* problem arises when parts of a system are modeled from different viewpoints and there is a certain overlap between the viewpoints. For example in UML, sequence diagrams and collaboration diagrams are both used for describing object interactions, from a temporal viewpoint and a structural viewpoint respectively. As a consequence there may be consistency problems between these two behavioral models.

The *vertical consistency* problem arises when a specification is transformed into another refined specification. It is desirable that the refined specification is consistent with the previous specification.

Various approaches for dealing with consistency problems, ranging from formal to algorithmic, have been studied.

### **Browsing and Navigation**

Automatic diagram layout is also a topic that is being looked at in the research community.

We are still in the process of surveying literature on the other topics we have proposed.

## Appendix A: State of the art in industry

### AllFusion Component Modeler

**Site** - <http://www3.ca.com/Solutions/Product.asp?ID=1003>

**Company** - Computer Associates

**Version** - AllFusion Component Modeler 5.0, released October 2003 (formerly known as Paradigm Plus)

**Overview** - AllFusion Component Modeler is a robust UML modeling tool for visualizing, designing and maintaining enterprise components for eBusiness. Through extensive support for collaborative modeling and component reuse, developers are able to share expertise and re-use components to eliminate redundant development efforts. AllFusion Component Modeler helps organizations deliver strategic multi-tier applications and evolve with today's eBusiness needs.

**Model Validation** - AllFusion Component Modeler has a module called the Model Xpert Engine whose purpose is model validation. Model Xpert provides tips, errors and warnings to guide users in developing projects according to the rules of UML and good design practices. Model Xpert supports two sets of rules: (1) the UML well formedness rules as they appear in the specification. (2) UML Model Xpert rules. The second set of rules are specific to AllFusion Component Modeler, and apply mainly to naming, typing, and referencing.

**Metrics** - Not supported.

**Anti-patterns** - Not supported.

**Browsing and Navigation** - Not supported.

**Version Difference** - Not supported.

### ArcStyler

**Site** - [http://www.io-software.com/products/arcstyler\\_overview.jsp](http://www.io-software.com/products/arcstyler_overview.jsp).

**Company** - Interactive Objects

**Version** - ArcStyler 4.0, released August 2003

**Overview** - ArcStyler is a cross-platform, pure Java, standards-compliant, seamless environment for the rapid design, modeling, generation, deployment and management of high-quality, industrial strength applications. The main technology breakthrough in ArcStyler's is the MDA-Cartridges that allow the capturing of platform specific knowledge for the purpose of model transformations and code generation. By "changing" cartridges the same model can be used for generating and deploying for different platforms.

**Model Validation** - Cartridges package all the technology-specific items and mechanisms required for transformations (automated model-to-model and model-to-infrastructure generation). This includes mapping and verifier engines.

**Metrics** - Not supported.

**Anti-patterns** - Not supported.

**Browsing and Navigation** - Not supported.

**Version Difference** - Not supported.

### iUML (Executable UML) iUML Modeler & iUML Simulator

**Site** - <http://www.kc.com/products/iuml/index.html>

**Company** - Kennedy Carter

**Version** - iUML release 2.0

**Overview** - iUML is a multi-user xUML(see definition bellow) design and simulation environment. It is built from the following top level pars:

- iUML modeler - visual xUML design environment that not only lets users to draw xUML diagrams in a way that maintains their integrity. This means that a user cannot build models that break the well-formed rules (through the usage of context sensitive pick-lists).
- iUML Generator - generates the code required for the model execution
- iUML Simulator - "Executable UML" - verify behavior of the models created in iUML by simulating its execution. Model execution, test and debug facility.

xUML - is a subset of UML incorporating a complete action language that allows system developers to build executable system models and then use these models to produce quality code. The xUML process is a rigorous OO system development method. It is based upon the principles of building a set of precise, testable, analysis models of the system to be developed, executing defined tests on these models, and defining a systematic strategy by which the models will be used to produce code for the desired target system.

Users define the logic using ASL (Action specification Language). The aim of the language is to provide an unambiguous, concise and readable definition of the processing to be carried out by a system. ASL is used to specify actions, operations, initialization sequences and test methods.

**Model Validation** - the xUML maintains integrity - conflicts are checked.

**Metrics** - Not supported.

**Anti-patterns** - Not supported.

**Browsing and Navigation** - Not supported

**Version Difference** – Supported

## NetBeans Metadata Repository (MDR) Project

**Site** - <http://mdr.netbeans.org/>

**Company** - open source project that is sponsored by Sun Microsystems

**Version** - NetBeans IDE Release 3.5.1

**Overview** - MDR implements the OMG's MOF standard based metadata repository and integrates it into the NetBeans Tools Platform. It contains implementation of MOF repository including persistent storage mechanism for storing the metadata. The interface of the MOF repository is based on (and fully compliant with) JMI (Java Metadata Interface - JSR-40). MDR also defines additional features that help to incorporate it into the IDE (e.g. event notification mechanism). MDR is rather an infrastructure project. To present the MDR functionality to the user it is usually necessary to write a module which uses the MDR.

**Model Validation** - Constraint checking mechanism included (meta-model)

**Metrics** - N/A

**Anti-patterns** - N/A

**Browsing and Navigation** - N/A

**Version Difference** - N/A

## Objectteering UML

**Site** - <http://www.objectteering.com/products.php>.

**Company** - Objectteering Software

**Version** - Objectteering UML 5.2.2, released October 2003

**Overview** - Objectteering/UML provides a complete, simple to use UML solution to analyze, design, implement, test and deploy software applications. At the cutting edge of

software development innovation, Objectteering/UML is the first UML tool to fully support the OMG's MDA™ approach. Objectteering/UML features top quality UML modeling facilities ensuring consistency between all UML elements, as well as high performance Java, C++, Corba IDL and SQL code generation, automated documentation generation for the complete project and teamwork facilities, through a multi-user repository.

**Model Validation** - Objectteering/UML denotes their well formedness checking "consistency checks". The tool provides over 200 such checks and they are done in real time, i.e. as the user creates the model. The checks are divided into two: obligatory checks which are essential to accurate modeling, and optional checks. Obligatory checks make sure that the modeling elements are used according to the UML meta model. Some examples: A type cannot have communication links; A parameter can have a class, a data type or an enumeration as its type; Only a class or a component can implement an interface. The optional checks are activated by default, but the user may deactivate them, and later check them manually, or turn them back on and fix any problems that are found.

**Metrics** - Objectteering provides a set of metrics which are used to evaluate the quality of the models produced. Parameters are measured on the model according to quality criteria fixed by users, such as complexity, testability, cohesion and stability. The Objectteering/Metrics module provides two types of metrics: counting metrics, and specific metrics. Some examples: Average number of attributes per class; the cohesion between classes in a package; the coupling between classes in a package; Depth of inheritance. Examples of specific metrics are: Class responsibility; Abstraction; Instability.

**Anti-patterns** - Not supported.

**Browsing and Navigation** - Not supported.

**Version Difference** - Not supported.

## Poseidon

**Site** - <http://www.gentleware.com/>

**Company** - GentleWare

**Version** - Professional Edition 2.0 \ Enterprise edition

**Overview** - Poseidon for UML is directly based on ArgoUML (open source project - version 0.8 - [www.argouml.org](http://www.argouml.org)). It comes with all UML diagram support, code-generation and reverse engineering capabilities. It's extensible (plugins) and has various offerings.

**Model Validation** - (in beta release only) the tool contains a built-in critiques auditing mechanism that includes critiques that check whether the model is as well-formed. The set of critiques that are checked is extensible.

**Metrics** - Not supported

**Anti-patterns** - Not supported.

**Browsing and Navigation** – Partially supported. You can search for an entity and see in what diagrams it is used. Refactoring is supported

**Version Difference** – In Enterprise edition only. Supports a collaborative environment where members of a development team can work on the same project concurrently. The project itself will be managed and stored at the server, while modifications of the model by client will be deployed to all the affected clients immediately. All clients will be kept in sync. with the server. Features like locking parts of the models for exclusive use and conflict checking are provided too. No information was found on how the tool visualizes (it at all) conflict/modification and on the checking mechanism.

## Rhapsody

**Site** - <http://www.ilogix.com/products/rhapsody/index.cfm>.

**Company** - i-Logix

**Version** - Rhapsody 5.0, released September 2003

**Overview** - Rhapsody is an MDD environment based on UML for the realtime and embedded market.

**Model Validation** - In addition to checking UML models for completeness and consistency it offers executable validation of the model. Developers can analyze and execute their models to test and debug intended behavior and functionality. The Sequence Diagram Comparison tool enables the user to perform comparisons, for example between hypothetical and actual message sequences.

**Metrics** - Not supported.

**Anti-patterns** - Not supported.

**Browsing and Navigation** - Rhapsody has a references feature that navigates from the definition of an element to places it is used in the model. There is also a search facility that supports both textual and model based searching. The Populate Diagram feature automatically populates a diagram with existing model elements. The user can select which model elements to add to the diagram and Rhapsody automatically lays out the elements.

**Version Difference** - Not supported.

## SDMetrics

**Site** - <http://www.sdmetrics.com>

**Company** – SDMetrics

**Version** - not specified

**Overview** - Metric tool for UML design measurement. The models are measured by an extensible built in set of metrics. Users can define new ones.

The inputs for the tool are:

- (Mandatory) An XMI Source File that stores the UML design you want to analyze
- (Optional) Meta-model Definition File (SDMetrics meta-model) that defines which UML elements SDMetrics knows about. The metamodel provides the basis for the definition of the metrics to be calculated.
- (Optional) XMI Transformations File that defines how the model elements in the SDMetrics meta-model are retrieved from the XMI source file.
- (Optional) Metrics Definition File that defines the set of metrics to be calculated for your UML design.

**Model Validation** - Not supported

**Metrics** - Supported. For entire set see  
(<http://www.sdmetrics.com/down/SDMetricsManual.pdf>)

**Anti-patterns** - Not supported

**Browsing and Navigation** - Not supported

**Version Difference** -Yes. Comparison of subsequent versions of a design (size metrics deltas quantify the growth in size of the design, Identify parts of the system design that have undergone much change, etc.) and comparison of design alternatives.

## Tau Generation 2

**Site** - [http://www.telelogic.com/campaigns/2003/global/taug22\\_launch/index.cfm](http://www.telelogic.com/campaigns/2003/global/taug22_launch/index.cfm)

**Company** - Telelogic

**Version** - TAU Generation 2 version 2.2, released October 2003

**Overview** - Tau Generation 2 is a UML modeling tool for realtime applications.

**Model Validation** - Supports on the fly model checking, marks the problems both in the model and in a list of errors and warning. The checks involve syntax checking of the text portion of the model, checking name binding, etc. Tau Generation 2 also has a Model Verifier module. This module verifies the behavior of the UML model to check that the implementation is correct. The module works with the code generated by the tool, and uses a model execution simulation to aid the user in stepping through the model. The simulation has three tracing methods: text trace, UML model tracking, and sequence diagram tracing. In the UML Model tracking, the user can view the different transitions and statements as they are executed in the UML state chart diagram.

**Metrics** - Not supported.

**Anti-patterns** - Not supported.

**Browsing and Navigation** - Supports model navigation in a feature called the Model Navigator. The navigator is a modal dialog that allows the user to browse and navigate through various aspects of any entity in a model. The Model Navigator has a number of different views allowing the cross-examination of the model based on the model's internal relations.

**Version Difference** - Supports comparing & merging versions originating from the same model. The comparison is done between model files (.u2 extension). Differences can be accepted automatically or manually.

## Together

**Site** - <http://www.borland.com/together>

**Company** - Borland

**Version** - Together Control Center 6.1, released June 2003

**Overview** - Together Control Center is an integrated collaborative development platform designed to simplify and accelerate the analysis, design, development, and deployment of enterprise applications. The groundbreaking Borland LiveSource™ technology in ControlCenter keeps software artifacts automatically synchronized so changes do not interrupt development.

**Model Validation** – Well formedness only.

**Metrics** - Together provides code audits and metrics as Quality Assurance features. Since Together supports full roundtrip, code metrics can be considered as model metrics. Audits check for specific rules to which the source code should conform. Together provides a wide variety of audits to choose from, ranging from design issues to naming conventions, along with descriptions of what each audit looks for and how to fix violations. Metrics quantify the code and highlight parts of code that need to be reworked.

**Anti-patterns** - Some of the audits can be viewed as anti patterns.

**Browsing and Navigation** - No special browsing and navigation features are advertised, though there is an automatic diagram layout feature.

**Version Difference** - Together supports the use of version control tools for code and model artifacts, but does not have a compare & merge feature specific for models.

## WayPointer

**Site** - <http://www.jaczone.com/product/features/>

**Company** - JacZone

**Version** - WayPointer 3.0

**Overview** - WayPointer is a non-intrusive tool to build software with the Unified Modeling Language (UML) and the Rational Unified Process (RUP). WayPointer continuously monitors the state of a UML model in Rational XDE or Rational Rose. Based on the model state and high-level goals, WayPointer offers proactive tips on how to best proceed with the development of the system. Also, WayPointer continuously monitors the model for

completeness, consistency, correctness and provides automated remedies with the click of a button, allowing for early and quick elimination of such problems.

**Model Validation** – the tool is built on top of XDE or Rose thus is compliance to UML. It includes additional model validations that make sure that the user doesn't break the analysis & design decisions (at different abstraction levels) by illegal relations. For example, if package "A" depends on package "B" then if the user will make a reference from a class from package "B" to a class of package "A" he'll be notified.

**Metrics** - not formally

**Anti-patterns** - not formally

**Browsing and Navigation** – N/A

**Version Difference** - N/A

## XDE

**Site** - <http://www-306.ibm.com/software/awdtools/developer/rosexde/features/>

**Company** - IBM

**Version** - Rational Rose XDE Developer Plus

**Overview** - Rational XDE is a complete visual design and development environment, offers software developers a model-driven development approach to building applications, including web-centric solutions. It allows users to work in a single environment, thereby avoiding the need to switch between numerous, different, none integrated tools. It is so flexible that it can be implemented three ways: alone via the included Eclipse IDE; installed into the IBM® WebSphere Studio™ Studio Application Developer and Integration Edition IDEs; or installed into Microsoft® Visual Studio™ .

It enables architects & designers to participate in model-driven development with UML. Developers can use architectural models and patterns as the basis for implementation, thereby accelerating the development of applications to conform to their architecture.

**Model Validation** - Validate whole or selected elements for compliance to UML and language-specific guidelines.

**Metrics** – N/A

**Anti-patterns** - Not supported.

**Browsing and Navigation** - basic support only. One can select an element and get all related classes etc. No query mechanism, no zooming or slicing.

**Version Difference** - Not Supported

## References

- [Allfusion Component Modeler 5.0 Help]  
Allfusion Component Modeler 5.0 Help, Topic "Interacting with Model Xpert"
- [Chiorean et al., 2003]  
Dan Chiorean, Mihai Pasca, Adrian Carcu, Cristian Botiza, Sorin Moldovan (Babes-Bolyai University, Cluj-Napoca, Romania), [Ensuring UML models consistency using the OCL Environment](#), *Workshop on OCL 2.0 - Industry standard or scientific playground? In UML 2003 International Conference on the Unified Modeling Language*, October 20-24, 2003, San Francisco, CA, USA.
- [Eichelberger & Gudenberg 2003]  
Holger Eichelberger and Jurgen Wolff von Gudenberg, "UML Class Diagrams-State of the Art in Layout Techniques", *VISSOFT 2003*, September 2003,
- [Engels et al., 2001]  
G. Engels, J. M. Küster, L. Groenewegen, R. Heckel, [A Methodology for Specifying and Analyzing Consistency of Object-Oriented Behavioral Models](#), In V. Gruhn (ed.): *Proceedings of the 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9)*, ACM Press, Vienna, Austria, September 2001, pp.186-195. ESEC/FSE Paper.
- [Eshuis and Wieringa, 2002]  
R. Eshuis and R. Wieringa. [Verification support for workflow design with UML activity graphs](#). In *Proc. International Conference on Software Engineering (ICSE 2002)*, pages 166-176. ACM Press, 2002.
- [Gardner, Griffin, Koehler, Hauser, 2003]  
Tracy Gardner, Catherine Griffin, Jana Koehler, and Rainer Hauser, [A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard](#), July 21, 2003
- [Gery, Harel, & Palachi, 2003]  
Eran Gery, David Harel and Eldad Palachi, [Rhapsody: A Complete Life-Cycle Model-Based Development System](#), I-Logix whitepaper
- [Gogolla et al., 2003]  
Martin Gogolla, Jörn Bohling, and Mark Richters. Validation of UML and OCL Models by Automatic Snapshot Generation. In Grady Booch, Perdita Stevens, and Jonathan Whittle, editors, *Proc. 6th Int. Conf. Unified Modeling Language (UML'2003)*, pages 265-279. Springer, Berlin, LNCS 2863, 2003.
- [Litvak, Tyszberowicz, & Yehudai, 2003]  
Boris Litvak, Shmuel S. Tyszberowicz, Amiram Yehudai, Behavioral Consistency Validation of UML Diagrams, *SEFM 2003*: 118-125
- [Moors, 2000]  
Michael Moors, "Consistency Checking", *The Rational Edge*, April 2000, <http://www.therationaledge.com/rosearchitect/mag/current/spring00/f6.html>
- [Objecteering/UML 5.2.2 Documentation]  
Objecteering/UML Modeler User Guide, Topic "Removable Consistency Checks"  
Objecteering/UML Metrics
- [Porres, Paltor and Lilius, 1999]  
[Ivan Porres Paltor](#), Johan Lilius, "vUML: A Tool for Verifying UML Models", [Automated Software Engineering, ASE'99, 14th IEEE International Conference, 1999](#)
- [Rational XDE 1.5 Help]  
Rational XDE 1.5 Help, Topic "Validating Models and Diagrams"  
Rational XDE 1.5 Help, Topic "Validating Java Models"
- [Richters and Gogolla, 2000]  
Mark Richters and Martin Gogolla. Validating UML Models and OCL Constraints. In Andy Evans and Stuart Kent, editors, *Proc. 3rd Int. Conf. Unified Modeling Language (UML'2000)*, pages 265-277. Springer, Berlin, LNCS 1939, 2000.

- [Schäfer, Knapp and Merz, 2001]  
*Timm Schäfer, Alexander Knapp and Stephan Merz*, [Model checking UML state machines and collaborations](#), *Electronic Notes in Theoretical Computer Science* Volume 55, Issue 3, 2001
- [Schmidt and Varró, 2003]  
Akos Schmidt and Dániel Varró, [CheckVML: A Tool for Model Checking Visual Modeling Languages](#), *UML 2003 International Conference on the Unified Modeling Language*, October 20-24, 2003, San Francisco, CA, USA.
- [Shen et al., 2001]  
[Wuwei Shen, Kevin Compton](#), and James K. Huggins, "A Toolset for Supporting UML Static and Dynamic Model Checking", *Proceedings of ASE2001 (16th IEEE International Conference on Automated Software Engineering)*
- [Telelogic Tau 2.2 Documentation]  
Telelogic Tau 2.2 User Documentation, Topic "Working with Models"  
Telelogic Tau 2.2 User Documentation, Topic "Verifying an application"
- [Together Control Center 6.1 Documentation]  
Together Control Center 6.1 User Documentation
- [UML Consistency, 2002]  
Research Report, First Workshop on Consistency Problems in UML-based Software Development, October 2002
- [UML Consistency, 2003]  
Research Report, Second Workshop on Consistency Problems in UML-based Software Development, October 2003
- [Varró, 2003]  
Dániel Varró, [Automated Formal Verification of Visual Modeling Languages by Model Checking](#), Accepted to the *Journal of Software and Systems Modeling, Special Issue on Graph Transformation and Visual Modeling Techniques*, Springer.