

UML for C Developers

Getting Embedded C Applications to Market Faster using the Model-Driven Development Technologies of Modeling, Simulation and Code Generation

Jerome L. Krasner, Ph.D.
April 2005

EMBEDDED MARKET FORECASTERS
American Technology International, Inc.

Embedded Market Forecasters
Research and Consulting
for Embedded Products,
Markets and Channels



Copyright 2005 by Embedded Market Forecasters, a division of American Technology International, Inc, 1257 Worcester Road #500, Framingham, MA 01701. All rights reserved. No part of this document covered by copyright hereon may be reproduced or copied without expressed permission. Every effort has been made to provide accurate data. To the best of the editor's knowledge, data is reliable and complete, but no warranty is made for this.

Table of Contents

<u>Overview</u>	4 - 5
<u>UML and Model-Driven Development (MDD)</u> <u>Deliver Value</u>	5 - 7
<u>The Special Concerns of the C Developer Using MDD</u>	7
<u>Model-Driven Development Productivity Enhancers</u>	8 - 12
<u>UML Based Modeling</u>	8 - 9
<u>Simulation</u>	9
<u>Production Code Generation</u>	9 - 10
<u>Model/Code Associatively (Roundtrip engineering)</u>	10 - 11
<u>Reuse of Legacy Code/ Reverse Engineering</u>	11 - 12
<u>Summary</u>	13
<u>Appendix</u>	14 - 17
<u>MDD for C Buyer's Checklist: Key Criteria when looking at UML based Model-Driven Development Environments</u>	15 - 16
<u>UML 2.0 Vendors</u>	16 - 17

Overview

Embedded designs have become increasingly more complex while the windows of opportunity continue to shrink. In the global world of embedded enterprise, it is not unusual to find OEMs and systems integrator design teams spread out across geography, operating in different time zones, experiencing overlapping areas of responsibility and finding bugs much too late in the design cycle.

The resulting design delays are expensive (EMF data shows a consistent 4-month average delay with 56% of all embedded designs completed behind schedule). 11% of embedded designs are cancelled – and the average time-to-cancellation is nearly 5-months. Engineering time associated with such delays and cancellations is very expensive – and does not include missed opportunity costs.

Annual surveys by Embedded Market Forecasters (EMF) of embedded developers has clearly shown that software development is responsible for more than 80% of design delays and associated design complications. This data also reports on embedded developer responses to design complexity. When asked how close their final design was to pre-design expectations (for performance, systems functionality, features and schedule) over 33% of respondents indicated that their final design was NOT within 50% of the pre-design expectation.

Respondents then indicated what steps they take if the final design is unacceptable. The five most mentioned actions either involved extensive reengineering or removal of systems features (these will be detailed in the EMF report “2005: Embedded Hardware/Software Design Preferences”).

As software developers sought out solutions to their development angst, they began to move to alternative design methodologies and tools that promised relief. These alternatives were largely embodied in object oriented (OO) methodologies with the latest and most advanced tools focusing on supporting the UML notation. UML and OO are largely unfamiliar to embedded software developers who are more accustomed to designing in a “C” language environment.

While OO software developers were becoming better able to address increasingly complex design requirements and shrinking development cycles, by taking advantage of the UML based modeling, simulation and code generation technologies available in the leading tools, the C software developers were forced to either embrace the unfamiliar OO environments or to be left out.

This exclusion is most strange as more embedded developers write in C language than in any other. Figure 1 presents a listing of programming languages currently in use by embedded developers, and those planned to be used in 2005.

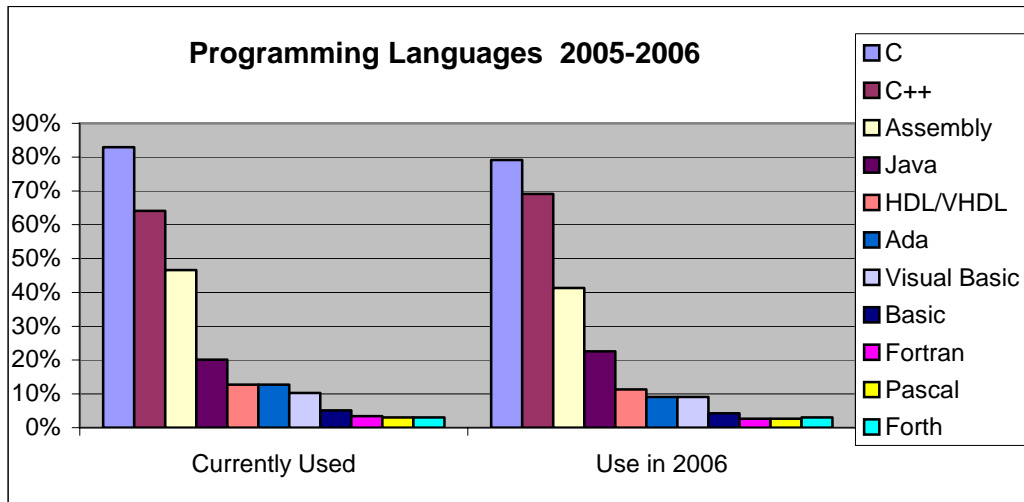


Figure 1: Programming Languages used by Embedded Developers

There are other reasons why the C language developers have not been able to adopt tools supporting modeling, simulation and code generation based on UML. Many past projects have been written using C and thus there is a large amount of legacy intellectual property (IP) that needs to be maintained and reused in future projects. The skill transfer from a C development environment to an OO environment is a major challenge and is being resisted by many C developers. On the other hand the complexity of C applications has been exponential driving the need for these advanced technologies. C developers increasingly express a need to understand and architect the design of their applications.

In this report, data is presented to highlight the need for UML-based technologies, including modeling, simulation and automatic code generation, to overcome the challenges facing today's embedded C developers, enabling them to get their applications to market faster. Furthermore an analysis of how these technologies should be extended to fit the needs and remove barriers to adoption is also detailed.

UML and Model-Driven Development (MDD) Deliver Value

Whether the system is poorly conceived, specified or whether crucial algorithms fail to adequately address systems performance, traditional methods of embedded software development are yielding to a process known as "Model-Driven Development (MDD)". MDD is used to more clearly analyze requirements, define design specifications, test systems concepts using simulation and to automatically generate code for direct deployment on the target hardware. (See EMF paper: "Reducing OEM Development Costs and Enabling Embedded Design Efficiencies Using the Unified Modeling Language 2.0")

One of the major advances in software engineering design has been the use of the Unified Modeling Language (UML)TM for enabling embedded design efficiencies. Pioneered by such companies as IBM/Rational and I-Logix for embedded applications, notwithstanding its value to addressing very complex designs and (with certain commercial offerings) the ability to go from diagrams (including Statecharts) to source code (automatic code generation), the technology has been providing major benefits of shorter design cycles and high quality products to those developers using OO.

EMF data has shown that:

- The use of simulation-modeling tools by embedded developers has reduced design delays and cancellations.
- The use of simulation-modeling tools by embedded developers has significantly improved the relationship between pre-design expectations and final designs.
- UML is the most popular graphical representation for simulation-modeling tools for discrete embedded system designs.
- IBM's acquisition of Rational Software has significantly broadened the use of UML for embedded applications.
- UML, simulation and code generation enables faster design iterations that produce desired performance, functionality and capabilities.
- Using UML, simulation and code generation, design cycles are more predictable and result in faster product shipments with lower project risk.
- UML, simulation and code generation contributes significantly to a reduction in design, development and implementation costs.

There are important aspects of this data that need to be emphasized. Final design/pre-design expectations are usually based on software development.

- 35% of embedded developers use simulation-modeling in their design practices, but only 11.5% also use automatic code generation with rapid prototyping.
- Of the developers that use automatic code generation in their design practices, 58% use it in conjunction with simulation-modeling (34% use it in conjunction with rapid prototyping).
- Simulation-modeling is largely a systems-based tool and is less frequently used as a software development tool. Enhanced systems design is an essential component of good design practices.
- From the 2005 EMF embedded developer survey: Systems Engineers - 44.7% use simulation modeling (25.4% for software engineers), 15.3% use rapid prototyping (7.9% for software engineers) and 12.0% (11.6% for software engineers) use automatic code generation in their designs.
- Constrained environments are still prevalent and are expected to remain so throughout the decade and beyond as simple connectivity devices emerge to meet the availability of higher and cheaper bandwidth. Although 32-bit designs remain the most used architecture (69.4% of responding developers), 8-bit (29.8%) and 16-bit (24.8%) designs have not decreased in number. To the contrary, it is clear from the multiple responses that 32-bit developers are incorporating 8-bit and 16-bit architectures into their designs.
- The number of new embedded designs per developer has increased slightly in the 2005 survey over the 2004 survey. Approximately half of new embedded designs are "new to the world"; the other half being a re-design of an existing product. Hence legacy code upgrades can be seen to constitute half of all designs. The ability to incorporate legacy software in a new UML design is clearly a very important feature to embedded developers.
- Legacy code was typically written in C even if the developers have moved on to C++ for newer projects.

The EMF 2005 survey is responded to by a broad cross-section of embedded developers and applications. Hence the numbers reported herein are significant.

The Special Concerns of the C Developer Using MDD

Even though the concept of MDD can solve many pains through visualization of requirements, model simulation and down to production quality code from the validated model, there are still some holes within UML and MDD that have been concerns for wide adoption by C developers.

The first concern is that the C developers' natural instinct is to think about the problem in a functional manner whereas most UML and MDD tools force the developer to think and design in an OO environment. Many C developers resist this transition from a C development environment to an OO environment. In cases where the C developer accepts OO environment, the transition often causes increased ramp-up time for the project.

The second concern is found in the importance that external code plays in the application. Almost all C development today contains some form of external source code. This code may be some special IP that needs to be maintained and reused. This could be existing source code that via reuse will give the project a head start or it could be code purchased from a third party that specializes in developing code for a specific application. The same concern is valid about having to include code generated from a domain specific modeling tool. For most of these cases reverse engineering the code into a model is not an acceptable option.

The third concern is the ability to directly control the generated code. This concern often comes up when a project has real-time concerns or if an algorithm needs to be coded in just the right way to maintain fidelity or if the underlining hardware needs to be directly accessed for performance.

The fourth concern is an ability to debug and fine-tune the actual code running on the target hardware and ensure that all the changes are dynamically reflected in the model.

The fifth concern only applies to C developers targeting highly constrained hardware environments that require very efficient code. In this case they need to be assured that the size of the code will not be greater than the code they would generate by hand or else risk an increase in recurring hardware costs.

With one-exception UML vendors, notwithstanding their efforts to generate C code from the OO UML designs, have not provided the capabilities that would enable C developers to exploit a MDD based process. Currently only I-Logix' Rhapsody provides such an environment by allowing C developers to work in either a Functional, Structured or OO environment and also enabling them to seamlessly integrate their existing code into their current application.

Model-Driven Development Productivity Enhancers

MDD contains technologies that are designed to drastically increase the productivity gains of one's organization and at the same time producing higher quality systems. The following sections will summarize these capabilities and the benefits brought by each. In addition special attention will be paid to the capabilities that specifically address the special concerns of the embedded C developer.

UML Based Modeling

Benefit – Modeling provides a clear understanding of the design intent and eases communication between team members and customers. In addition it makes team based design easier due to a graphical means of partitioning the design and defining interfaces.

UML 2.0 is a standard set of graphical notations that enables Model-Driven Development (MDD). The graphical notations in UML 2.0 for performing design architecture are based on OO techniques. When developing C applications the OO ability to define completely encapsulated elements enable developers to control complexity as the design scope increases.

The level of comfort, skill transfer and risks associated with transferring from a C development environment to an OO environment is a major challenge and has been resisted by most C developers. A MDD tool supporting the C developer can ease the transition from textual programming to graphical modeling by allowing C developers to work in an intuitive manner in an environment natural to them. C developers typically think in a functional manner as opposed to an OO manner. By extending UML and providing functional modeling constructs based on the C language in addition to the OO constructs allows modeling in a functional manner. Adding the C constructs of *files*, *functions* and *variables* into the graphical environment makes it easier and more intuitive for C developers when constructing models. It is important that the other MDD productivity enhancers including simulation, production code generation and reverse engineering support these constructs. Using the natural C constructs still brings the benefits of clear design intent and eased communication while enabling team based design.

Some C developers prefer to follow a structured modeling paradigm instead of an OO method. By providing graphically the concepts of Blocks and Flows, which dominate the typical structured methodologies, C developers are quickly able to adopt a UML based MDD environment without having to learn a new methodology. In order to provide the most gains to the C developer, UML tool vendors must support these concepts.

For the C programmer who prefers to work in an OO environment it is important that the MDD tool supports the OO concepts of Class and Object that UML provides for. This allows these developers to work in a similar environment to what the C++ and Java developers who are leveraging MDD to their benefit.

By providing these different design methods and notations the C developers can chose the one most intuitive to them. This overcomes the difficulties of skill transition that occurs when someone is forced to learn and use a new methodology. It also eliminates any risks associated with switching to designing with UML based models for C applications. In addition providing a mechanism that allows users to migrate from one method to another or to even work with multiple methods within one design can ease the transition from textual based design to UML model-driven development.

C developers have one other challenge that is often not faced by the C++ developer and that is the need to get the code to look exactly the way they need it to. This can be due to a number of reasons such as having an algorithm that needs to meet very tight timing requirements or the need to directly access the hardware to ensure that registers are properly accessed. The only way to do this is to let the developer write the C code directly for these instances. Some tools force the developer to use propriety language syntax for specifying actions, which is counterproductive for C developers.

Simulation

Benefit - Allows validation of the model at any point in development. This enables the C developer to find and eliminate errors early in the process when they are least expensive to fix.

Constant validation of the system is key to producing high quality systems very quickly eliminating the need to wait until all the design and implementation is done and prior to testing the model. The closer the model execution resembles how the model will execute on the target system, the greater the value provided by MDD.

One aspect that C developers often will need to analyze is the sequence of various functions that are called from one design entity to another. In UML a sequence diagram is used to show this interaction between elements. It is important that a MDD tool for the C developer will show the user how this interaction is occurring on the sequence diagram during simulation.

The other important point is to ensure that when simulating the modeled portion of the design any external source code can also easily be included in the simulation. This not only enables the integration of these pieces to be validated but also ensures that the desired results are achieved when the complete application capabilities are being used.

Production Code Generation

Benefit - Gets one to final product quicker with fewer defects and lower risks. It frees up C developers to work at a higher level of abstraction in order to quickly create even very complex systems. A lot of systems today have become so complex that handwriting the entire system would require years as opposed to months using MDD technology.

Generation of 80-90% of the final application by a good UML based MDD tool is a typical benchmark. However for a high quality system, the code must be readable, easy to understand and easy to customize. This is critically important when issues need to be dealt with in the field where the MDD tool is not in the debugging loop and source level debugging is the only tool available.

Most MDD tools force the C developer to model using OO constructs and as such these tools will usually produce C code that is convoluted and difficult to read due to the nature of mapping the functional based C code to the OO constructs. It is important for C development that the user can get code out that is readable and easy to understand and often times there is a need to make sure the code fits certain internal standards. These needs can be met by a combination of adding C concepts to the modeling language that can provide a more direct mapping to the generated code and by providing the user with the ability to customize the code. Typical and effective customization can be achieved either via properties, templates or rules based code generation.

It is also important that the code generator is capable of including C code created outside of the modeling environment so that legacy code, third party code and code that just needed to be written by hand can be included in the overall build and included while simulating the application in order to find and eliminate errors for the entire application.

With the increasing number of 8-bit and 16-bit designs that can benefit from UML and MDD it is important to have C code generation capability for constrained targets (Figure 2).

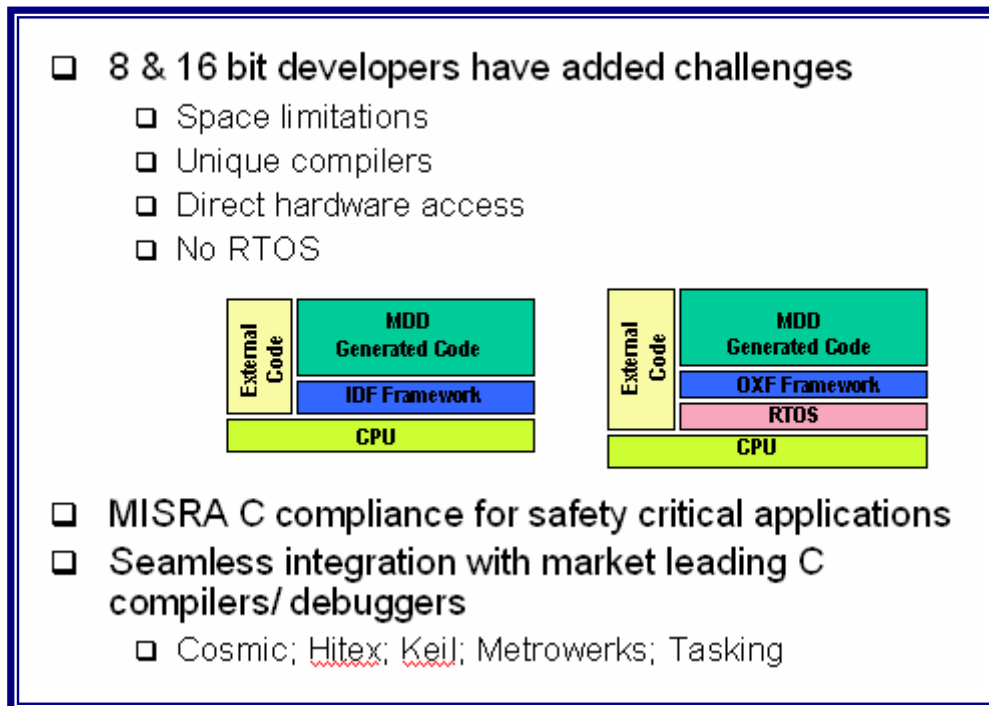


Figure 2: Code Generation for Constrained Targets

Model/Code Associatively (Roundtrip engineering)

Benefit - Code and model always stay synchronized so that the graphical representation always represents the end production code. This ensures that the code being executed on the production system is representative of the design and requirements of the application.

It is very important that the code being run on the production system is truly representative of the requirements and design of the system. This can be achieved in a UML tool by changing either code or model view and having the changes in one view reflect in the other. The ease of which this is done gives the developers a lot of flexibility in their design capabilities.

This is especially critical for C applications that have algorithms with difficult to meet real-time requirements or that require direct interaction with the hardware. Take a typical situation of being in the lab and realizing that the algorithm that was generated from the model does not remain stable. The developer needs to be able to fix this by going directly to the code to make changes. Now unless the UML tool has a strong Model/Code Associativity capability the developer is stuck with either having the code no longer match the model or she would have to go back to the model and try to change it in a way that would produce the code exactly the way it needs to be. Hence it is critical that the developer can make a change directly to the production C code and have the model automatically update. This ensures that the developer can get the code to be exactly as required while ensuring the model and the code are always synchronized.

Reuse of Legacy Code/ Reverse Engineering

Benefit - Allows the reuse of existing Intellectual Property (IP), saving considerable time when new designs are begun.

The ability to graphically represent existing code in a model makes UML based MDD very appealing to those organizations where it would be too costly to try and start designing a system from scratch.

It is also crucial that after reverse engineering the UML tool produces an equivalent system during the code generation process. The reason of course is to ensure that the existing code functions the same way when in the MDD environment.

C applications almost always have some form of existing code to deal with. Sometimes this code is used as a starting point and will be modified enabling the project to get a head start. Other times it will represent just a small piece of the project and will be brought in as is. An example of this might be when the code is generated from a domain specific tool or purchased from a third party. In other cases it will be the baseline code for the project in which some pieces will not change at all but others may be modified to enhance the functionality or to ease integration with new functionality. This presents a number of different challenges that MDD can help address providing the right capabilities exist.

One challenge is how the developer understands the C code that they need to deal with. Unless they wrote it, this can be a very daunting undertaking. A picture is worth a thousand words and a graphical representation of the code is always easier to read and understand than the source code itself.

Another issue is to ensure that when this code is integrated with the newly developed code it works. Being able to visualize the interfaces between the external source code and simulating or testing the external code in conjunction with the model that is being developed will greatly enhance the ability to integrate this existing code into one's new design.

Today most of the leading MDD tools force the developer to either convert this code into an OO UML model or treat the code as a black box. Both of these methods obviously still make it difficult to understand or to modify the code going forward and provide little value with integration. To truly take advantage of MDD requires the ability to either reverse engineer the code into a model or visualize the code externally. By combining these technologies with the natural UML language extensions of files, functions, variables, blocks and flows, C developers can easily include existing code into their applications in a manner that makes them easy to read, modify and test.

Visualization enables a graphical representation of the source code in the model diagrams. It is essential that this visualized code can be analyzed and integrated with the new capabilities that are created by modeling. The visualized code should however remain external to the model and remain untouched so that extensive unit testing on this known good code is not needed.

Reverse engineering of the existing code allows the C developer to bring the code into the model enabling it to be analyzed and integrated with the new code. It can now also be easily modified and simulated to ensure that any changes are correct and that they do not break anything else. It is important that this code be brought into the model in a manner that maps directly to how it was written and not be modified to map to a specific tool's graphical notational limits. Otherwise this modification will usually result in a model that is difficult to understand.

Quite often migration path from visualization method to reverse engineering is also critical. The developer starts off by visualizing the legacy source code and integrating it with the new features in the model and then wants to make major changes to the legacy portion of the code. In this case the developer probably would wish that they had reverse engineered this piece of code rather than visualized it externally. Hence the UML tool must be able to select the piece of the code needed to be changed and be able to reverse engineer this into the model as and when desired by the developer. Very few UML tools on the market provide this critical capability.

Summary

Embedded devices are becoming more wide spread and are increasing in complexity everyday. To deal with this designers of embedded systems are leveraging the benefits of UML based MDD. They are now able to find and correct errors earlier in the process when they are less costly and to develop the systems more rapidly and with higher quality and lower project risk

In this paper EMF has shown that the evolution of Model-Driven Development (MDD) now requires that C developers' unique needs be met by UML development tools in order to insure that final designs are consistent with pre-design expectations.

- C developers have unique needs for Model-driven development (MDD)
 - Functional orientation
 - Graphical modeling of C language concepts e.g., files, functions, variables etc.
 - Flexible reuse of legacy code
 - Code generation for constrained environments
 - Need to work close to hardware
- C developers face the same challenges of complexity, productivity and time to market as OO developers

Rhapsody in C, from I-Logix brings the advantage of UML based MDD, in a natural and intuitive way, to the C developers.

EMF expects that other UML vendors will respond to the needs of this major market opportunity to address the needs of C embedded systems developers and to integrate them into the design team.

APPENDIX

MDD for C Buyer's Checklist: Key Criteria when looking at UML based Model-Driven Development Environments

UML 2.0 Vendors

MDD for C Buyer's Checklist: Key Criteria when looking at UML based Model-Driven Development Environments

Criteria: Works with Legacy Systems
Visualization of legacy code (work with external code "as is" with a visual representation in the model)
Reverse Engineering of legacy code (builds/populates code information so its viewable in the model)
Auto synthesizes diagrams from Reverse Engineered code (allows user to pick what they want auto synthesized)
Can work at the code level and have round trip engineering ensure code and model are in sync manually or dynamically
Ability to reference legacy artifacts in tool (such as code), without explicitly having to import legacy
Can import legacy models through XML (or other interface)
Criteria: Simulation
Model level debugging (run time visualization of state machines, activity diagrams, sequence diagrams, etc)
Model execution supported in all design phases (requirements/analysis, design level execution, auto test execution)
Can simulate one piece of the system
Can simulate the entire model at once
Can simulate on the target
Parallel execution of model level and optional source code level debugging (with source level debugger in the loop)
Criteria: Production Code Generation
Complete behavioral code generation from models for the target environment (not code frames)
Seamless retarget of generated code to appropriate RTOS environment
Can be targeted for custom RTOS environment
Ability to directly use the RTOS services
Customizable for smaller devices or to run with no-RTOS (8/16 bit applications)
Rules-Based Engine to enable customization of the generated code
Ability to include/exclude model components for compilation when generating code from models
Criteria: Requirements
Ability to capture non-functional requirements
Ability to link non-functional requirements to the model
Can generate requirements traceability matrix from requirements in model
Can import non-functional requirements into MDD tool (from Excel, Word, or a Requirements Management (RM) tool)
Can export non-functional requirements to RM tool.
Criteria: Analysis:
UML extensions for C concepts (files, functions and variables)
Key UML 2.0 Modeling Capabilities (Ports, Information Flows, Structured/Composite Classes, Sequence Diagram Enhancements, Activity Diagram Enhancements)
Ability to model distributed systems and allocate software artifacts to deployment nodes
Link analysis model to requirements for traceability
Ability to capture functional breakdown of system
Model checking for model completeness and consistency
Criteria: Design:
Can be adaptable to any design process (spiral, waterfall, V)
Ability to decompose analysis models with design artifacts
Modeling of multithreaded and multi processor environments
Hyperlinking of Design model to Analysis model to show traceability and navigability

MDD for C Buyer's Checklist: Key Criteria when looking at UML based Model-Driven Development Environments

Criteria: Test
Automated model based testing through execution on host or target
Command line test interface for scripting for execution of regression tests
Requirements-based (use cases) validation capabilities (use case and scenario execution)
Auto Test Generation capability
Test Case traceability via test management tool interface
Criteria: Openness
Off the shelf integrations with 3 rd party Requirements, Configuration management, and Testing.
Standard XMI interface
Tool API to allow integration possibilities with other tools
Criteria: Collaboration
Visual differencing and merging to enable parallel development
Graphical comparison of diagrams
Web interface for online design reviews
Automated documentation capabilities for design publishing/review
Integration to Requirements tools and bidirectional navigation (DOORS, RTM, RequisitePRO, etc.)
Criteria: Vendor
Tool training courses at customer site
Advance tool training classes
Support
Consulting services
UML training (non tool specific)
Audit services/Best Practices
Web site for useful information
Client freeware site
User group

UML 2.0 Vendors

ARTiSAN and Real-time Studio are trademarks or registered trademarks of Artisan Software Tools Ltd. All other trademarks are the property of their respective companies. www.artisansw.com

Rational and RoseRT are registered trademarks of IBM in the United States; all others are trademarks or common law marks of IBM in the United States. www.IBM.com/rational

Rhapsody[®] and I-Logix, among others, are registered trademarks and/or registered service marks of I-Logix Inc, in the United States and other countries. www.ilogix.com

Telelogic, Telelogic DOORS, Telelogic DocExpress and Telelogic TAU are the registered trademarks of Telelogic. Telelogic TAU/Architect, Developer, Tester, SYNERGY and ActiveCM are trademarks of Telelogic. All other trademarks are the properties of respective holders. www.telelogic.com

UML and UML 2.0 are registered trademarks of Object Management Group, Inc. in the United States and/or other countries.

Embedded Market Forecasters
Research and Consulting
for Embedded Products,
Markets and Channels

