Reducing OEM Development Costs and Enabling Embedded Design Efficiencies Using the Unified Modeling Language (UML 2.0)

Getting to Market Faster with Lower Cost

Jerome L. Krasner, Ph.D.
February 2004

EMBEDDED MARKET FORECASTERS

American Technology International, Inc.

Table of Contents

Table of Figures

This page intentionally blank

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 by 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 completely 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 complications. 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 (see EMF report “2003: Embedded Hardware/Software Design Preferences”).

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 define design specifications, test systems concepts and to automatically generate code for rapid prototyping as well as for software development.

One of the major advances in software engineering design has been the use of the Unified Modeling Language (UML)™ for enabling embedded design efficiencies. Pioneered by companies 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 Statecharts to source code (automatic code generation), the technology suffered due to incompatibilities of the various commercial UML products. Despite the fact that there was a UML standard, vendors offered their own extensions and variations that resulted in UML incompatibilities.

In order to rectify the problem of scalability and to make UML more available to OEMs, embedded developers and systems integrators, the major vendors and users came together to create the UML 2.0 standard.

From a commercial viewpoint, vendors that support the UML 2.0 standard need to find ways to differentiate themselves from their competitors while maintaining their support.

In this report, data is presented to demonstrate that UML-based technologies, including simulation-modeling, rapid prototyping, hardware-in-the-loop testing and automatic code generation, offer better design results with considerable savings to OEMs, embedded developers and systems integrators.

An analysis of how the major UML vendors differentiate themselves while maintaining support for UML 2.0 is also presented.

Looking at Embedded Design Processes

The following information was developed from the Embedded Market Forecasters 2003 Survey of Embedded Developers (www.embedded-forecast.com). The results presented are consistent from surveys conducted over the past 24 months.

Figure 1 presents a summary of design results according to schedule. The results are also cross-tabbed according to architecture and vertical markets.

Percentage of Design Completions According to Schedule				
	Ahead	Behind	Cancelled	Outsourced
Total Response	15.5%	54.0%	13.1%	11.6%
8-bit	17.1%	53.7%	13.7%	11.3%
16-bit	18.0%	49.5%	12.6%	12.5%
32-bit	15.4%	53.5%	13.1%	11.4%
64-bit	17.7%	47.3%	11.9%	16.8%
DSP	17.2%	54.8%	12.9%	11.3%
FPGA	15.8%	54.3%	12.6%	11.0%
Auto-Transport	14.8%	55.4%	15.1%	13.1%
Avionics	15.1%	52.0%	11.8%	15.2%
Bus Mach & Peripherals	15.1%	52.9%	14.8%	11.6%
Consumer Electronics	17.2%	52.3%	14.8%	11.9%
Datacom	11.8%	57.2%	16.5%	13.2%
Telecom	10.6%	60.2%	18.3%	7.5%
Electronic Instrumentation	18.3%	57.3%	13.3%	11.1%
Industrial Automation	19.1%	51.3%	13.0%	8.1%
Medical	18.1%	56.2%	11.6%	11.3%
Military	17.6%	52.1%	8.3%	14.3%

Figure 1 - Percentage of design completions according to schedule

The cost of delays and cancellations were established in an EMF report *"2003: Embedded Hardware/Software Design Preferences"* which took into account the number of developers per project, the average cost per developer and the period of delay (be it a delay in design completion or the period between design start and cancellation). By assigning a cost per developer and knowing from the data the average number of hardware and software developers per project per architecture and per vertical market application, the cost of delays and cancellations can be calculated by assuming an average cost per developer.

For example, it would not be unusual for the cost of delays of an Avionics application development to be in the range of $50,000 to $300,000 per month and the cost of cancellations to be in a similar range (excluding the cost of purchased tools and machinery no longer of use). For larger Avionics undertakings an order of magnitude increase in costs could be expected.

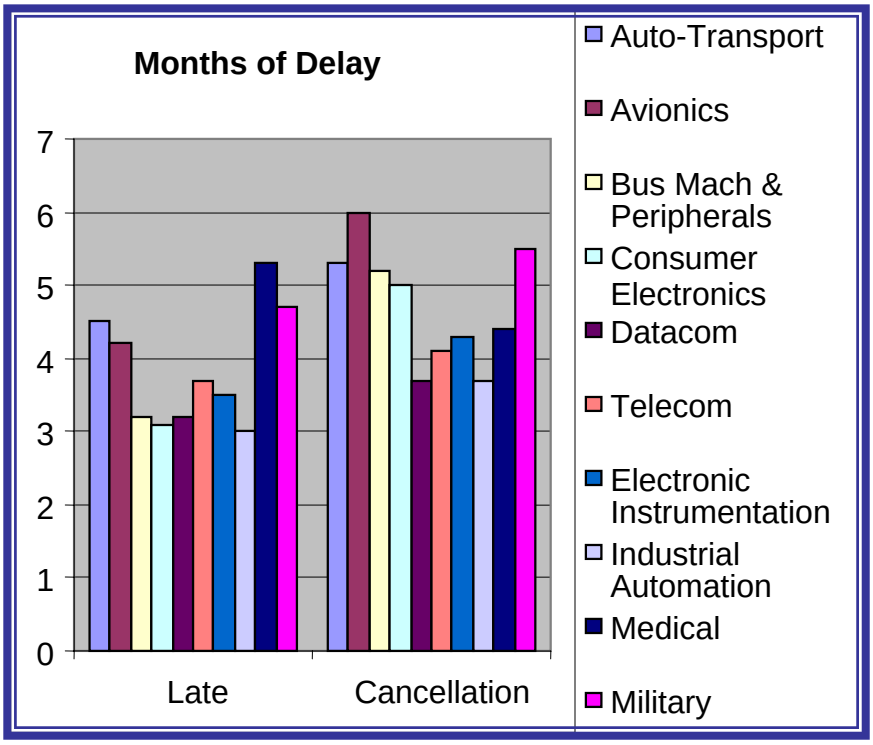

Figure 2 - Months of Delay for late completion or cancellation

Of further concern to an OEM is the relationship between final design results and pre-design expectations. Failure to approximate pre-design expectations can result in design delays, removal of product features and other non-productive actions.

In a survey of 947 embedded developers, each was asked to reply to the question, "How close to your pre-design expectation was your final design?" Three categories were presented, Performance, Systems Functionality and Features/Schedule. Developers were given the following choices: within 10%; within 20%; within 30%; within 40%; within 50%; and *not* within 50%.

In such a case a developer responding, say, to performance with "within 30%" was indicating that the design was within 30% but not within 20%.

Figure 3 presents the results from the 2003 EMF embedded developer survey.

Comparing Final Design Result to Pre-Design Expectations

	Within 10%	Within 20.0%	Within 30.0%	Within 40%	Within 50%	Not Within 50%
Performance	31.7%	19.3%	8.7%	2.3%	6.7%	31.3%
Systems Functionality	39.0%	14.0%	6.6%	1.9%	5.8%	32.8%
Features & Scheduling	19.8%	18.6%	13.8%	5.4%	15.8%	26.6%

Figure 3 - Comparing Final Design Result to Pre-Design Expectations

Cross tabulations were run according to 10 vertical markets, 7 microprocessor families, bus architecture, microprocessor architecture and 5 types of engineers.

Over the preceding two years of surveys, the comparison continues to worsen. It is clear that with over 30% of designs failing to approximate 50% of pre-design expectations (with 40% not being within 40% of expectations) that as designs become more complex, OEM design expenditures will increase while the window of opportunity and product features will be challenged.

Cross tabulations were conducted to compare final design results/pre-design expectations between all respondents and those that use simulation-modeling, rapid prototyping and automatic code generation in their designs.

EMF data also 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 enables faster design iterations that produce desired performance, functionality and capabilities.
- Using UML, design cycles are more predictable and result in faster product shipments.
- UML 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 2003 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.

- The EMF 2003 survey is responded to by a broad cross-section of embedded developers and applications. Hence the numbers reported herein are significant.

UML 2.0 to the Rescue

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 many holes within MDD that lie within the visual modeling infrastructure used to design systems. One major area of concern is that there hasn't been a modeling language that can correctly and satisfactorily address the needs of the systems and software engineers, so that a systems level design can seamlessly be taken down through software design and detailed sufficiently enough for full production code generation.

UML 2.0 is the visual modeling infrastructure intended at bridging the gap between systems and software, by providing new diagrammatic elements within its superstructure that can be used to sufficiently describe the system in enough detail for systems and software engineers. This in itself does not solve the problem, but does provide the necessary infrastructure so that an environment can be created to bring life to the infrastructure through execution and validation of the system, translated to production quality code for the intended target system.

Specifically what does the UML 2.0 bring to enable sufficient systems and software engineering? UML creates a framework for improvements in static, dynamic and behavioral diagrams for large-scale complex systems where systems' engineering plays a crucial role.

Let's look at the major innovations in static, dynamic and behavioral diagrams covered in UML 2.0.

Static Diagrams

The major innovations in static diagrams are with the additions of Structured Classes, Ports and Information Flows. Focusing initially at the most abstract to more concrete, Information Flows are an important tool for systems engineers to define the flow of data between architectural pieces of the system. On the Information Flow itself, the systems engineer can define Flow Items, which describe the actual data that flows in and out of the architectural components of the system. These diagrams can be further detailed directly by software in many ways. For example the Flow Items on the Information Flow can be realized by the software designers as actual asynchronous or synchronous messages, or even some complex data structure. The architectural components on either end of the Information Flow can be realized as UML 2.0 Structured Classes and decomposed into more manageable parts. A Structured Class allows you to take something that is more comprehensive in nature (such as the system or subsystem) and build components inside of it (called Parts), which are parts of its internal organization that make up its architecture. Systems and software teams can perform this decomposition, while enabling hardware and software design trade-offs to be made.

The level at which Structured Classes can be decomposed is theoretically infinite allowing detailed granularity of the system. For example a systems engineer may create sub-subsystems to further define the overall architecture and then a software engineer

could further detail those sub-systems into the concrete algorithms or software components.

An Information Flow and the items on the flow line describe the data coming into and out of an architectural component of the system. To realize the actual interface of the component, UML 2.0 Ports can be used to define the exact messages that come into and go out of a component at a particular point. A component can have many Ports (or one can think of them as entry/exit points) with the overall purpose of the Port is to encapsulate the subsystem by providing clean inputs and outputs through its Port interfaces. Interface definition is crucial to systems and software engineering alike as it allows subsystem developers to independently create the architectural details of the subsystem without having to worry about the details of the rest of system, enabling concurrent engineering. Adherence to the Port interfaces is what is important.

Dynamic Diagrams

The main focus of dynamic diagrams is to capture the relevant collaborations among the system components. This is primarily done through Sequence Diagrams, which not only show how subsystems communicate during run-time based on a particular system scenario, but also how the internal parts of the system interact with one another. However, with UML 1.X, Sequence Diagrams proved to be one-dimensional in nature and had difficulty in capturing large-scale system interfaces. Where there are potentially thousands of collaborations. Sequence Diagrams ended up not being a viable solution. Thus those systems designers lost the benefit of being able to graphically view how their system should behave at run-time. UML 2.0 has solved this deficiency at both the systems and software level by introducing the concept of Interaction Occurrences and Lifeline Decomposition.

Many times it is advantageous to capture a common scenario between a set of lifelines and just refer to that scenario rather than having to redraw the collaboration. This is especially important in systems where potentially thousands of collaborations need to be captured. In order to handle these scenarios graphically, there must be a way to organize sequences in such a way where one or many scenarios can be reused within larger scenarios. The Interaction Occurrences added to UML 2.0, such as the Reference Occurrence allows you to simply reference collaborations between a set of lifelines. The Interaction Occurrence is a new symbol added to Sequence Diagrams, which can easily be added to scenarios to create and capture these more complex system interactions. The key is now these diagrams scale up to large complex systems such as those typically found within the Mil/Aero community.

Lifeline Decomposition allows you to easily decompose a lifeline on a Sequence Diagram, which could represent the system or a component of the system, into a series of parts, so that you can easily drill down from large pieces of the system to view the internal message collaboration between the internal parts of the system. A use case of this might be a designer capturing a structured class view in one Sequence Diagram while easily navigating to its internal parts represented in another Sequence Diagram. With UML 2.0 it is now possible to create this linkage through a simple decomposition of the lifeline.

Behavioral Diagrams

Statecharts, invented by Dr. David Harel, founder of I-Logix, was a crucial concept added to UML, which made it possible to capture state-driven reactive systems. Statecharts go beyond the typical Mealy and Moore state machine because more complex behaviors can be captured by the decomposition of states and through the modeling of concurrent states, so that it is feasible for reactive pieces of the system to handle multiple behaviors concurrently. The introduction of Statecharts into UML has made it possible to use UML to model real-time reactive systems.

UML 2.0 presents another level of flexibility in defining complex systems behaviors, through Statechart inheritance. Statechart inheritance allows you to easily reuse existing behavior by enabling you to capture a common set of behaviors into a component of the system. These can be extended or specialized by other system components rather than defining the behavior from scratch. This means reuse of behavior. The most important aspect of object-oriented technology is the ability to use generalization, so that components can be reused by extension or specializations, thereby making software reuse a reality. This same concept is now applicable to UML 2.0 so that state behaviors in Statecharts can now be reused and extended through graphical notations.

Going Beyond the Standard

There are two clear business needs expressed by customers of Model-Driven Development and UML before its acceptance can become became widespread. One is the need for a single integrated development environment eliminating the ‘manual hand-off’ gap between Systems Engineers and Software Engineers. Secondly, the need for UML to be applicable to engineers that are not following an object-oriented method as well as those that follow an object-oriented method.

UML 2.0 only partially addresses the critical need for interaction between systems and software engineers. Companies are increasingly looking for a single tool that can enable both disciplines to work in one integrated environment. A competitive market segment is a healthy one that promotes growth. So how do UML vendors differentiate themselves to create new usages by embedded developers (which contributes to growth) while adhering to the UML 2.0 standard?

Using I-Logix as an example, they released Rhapsody 5.0, which extended UML to include some significant capabilities critical for systems engineers. For example, the ability to create Functional Block diagrams including data flows permit the systems engineer to work with familiar modeling notations, while remaining integrated in a standard UML 2.0 environment. A new Requirement Element was added, allowing textual capture of the system requirements that can be linked to other model elements. This ensures the system engineer has implemented and tested all the requirements of the system and ensures a tight cohesion between textual requirements and the system and software model. Lastly, smart Type Modeling assists the system engineer in defining data types, permits the use of classes as types and introduces language-independent types, thus increasing design portability and decreasing the complexity of design capture. Activity Diagrams, an essential view for systems engineers that details the functional flow of system activities, in Rhapsody have always been executable. With their release of Rhapsody 5.0 operations in Activity Diagrams can now be executed as well. This allows system engineers to observe and predict what will occur when a set of

activities is run. These new capabilities, for the first time, enable systems and software engineers to work in the modeling paradigm with which they are individually comfortable, yet all within a single integrated environment based on the UML standard.

Continuing the example, the needs of a C developer are quite different from those of a C++ developer. Rhapsody 5.0 includes features that enable more natural modeling for C development, by improving the efficiency of the C code that is produced. For example, the same Functional Block diagrams that were added for systems engineers also benefit the C developer, who is more apt to know functional modeling than object-oriented modeling found in UML. Many C programmers deal with resource-constrained environments, especially when using 8 and 16 bit devices. With Rhapsody 5.0, developers can select options to optimize the code size to reduce required ROM and RAM, enabling the selection of a lower cost microprocessor. Looking beyond the standard, I-Logix developed Rhapsody 5.0 to support options that allow the code to comply with the required set of guidelines specified by the Motor Industry Software Reliability Association (MISRA). This support enables engineers in the Automotive, Aerospace, Defense, Telecom and Medical industries who need to follow the MISRA guidelines to get the full benefits of Model-Driven Development (MDD). These benefits allow the C developer to work in the environment to which they have grown accustomed.

The UML 2.0 standard emerged as a result of the collaboration among UML vendors and users as well as from intellectual property contributed by vendors for the common good. By doing such the resulting UML 2.0 framework permits vendors to address a larger number of developers that can use the technology. By creating a larger market base, it is assumed that a larger market will better suit all vendors.

Rhapsody, as RoseRT and other UML-based solutions has had for some time supported a number of the concepts now defined in UML 2.0 with respect to architectural modeling. I-Logix, for example, contributed to UML 2.0, Statechart Inheritance that enables easy reuse of the behavioral parts of the model, Structured Classes that are composite classes with parts provides for easy hierarchal decomposition and information flows, which they have supported for a number of years.

By making such contributions and by releasing UML offerings with features consistent with UML 2.0 but with greater design flexibilities, I-Logix, IBM/Rational and Telelogic, among others, have sought to differentiate themselves while contributing enhanced design capabilities to OEMs, embedded developers and systems integrators.

UML 2.0 Tools of Importance

Let's consider the major tools of importance that UML 2.0 brings to embedded developers.

- **Ports** – Ports can be attached to model elements in order to define the required and provided interfaces of the element (the set of allowable inputs and outputs). The benefit of this is that system and software designers can easily encapsulate model elements by defining the access of it through the Port. This enables independent subsystem development as well as component based development.
- **Sequence Diagrams** – Enhancements in UML 2.0 were made for large system scalability. The two main enhancements were in the addition of interaction occurrences to describe how a particular sequence of events occurs through operators and the second addition was lifeline decomposition. Lifeline decomposition is the ability to decompose a piece of the system into parts, making it easier for designer to drill down from the top level of the system down to finer levels of granularity.
- **Information Flows** – A systems engineering enhancement to UML 2.0, Information Flow, allows designers to describe the data flow between system components. This adds to the understanding of system component communication and helps give a better understanding of how the component interfaces will be realized (which can be done through Ports). The data or flow items can later be realized in software development by actual concrete messages or data structures.
- **Structured Classes** – These were added so that larger pieces of the system such as components or subsystems can be easily broken down into more manageable parts through decomposition. This provides a clear link between larger pieces of the system and how it's broken down into software elements.
- **State Machine Inheritance** – This was added to reuse state machine behavior through generalization, so that more complex behaviors can be captured for more complex systems. This is also very useful for simpler systems to allow the reuse of simple behaviors that can be expanded in a variety of ways with Statechart notations.
- **Activity Diagram enhancements** – UML 2.0 Activity Diagrams are useful to model system level behavior of use cases and other system components (subsystems, components, classes). Activity Diagrams are similar to flow charts and data flow diagrams of structured methods.

How Vendors Differentiate Their Tools While Supporting UML 2.0

There are four major vendors for UML 2.0 – I-Logix (Rhapsody), Telelogic (TauG2), ARTiSAN (Real-time Studio) and IBM/Rational (RoseRT). For purposes of discussion, EMF chooses to use these four to show how they differentiate their offerings while remaining faithful to the UML 2.0 standard.

- **Ports** – Ports for both Telelogic TauG2 and RoseRT require one to use signals or asynchronous messages on the port interface. There is no way to define just simple synchronous operational behavior on the Port interface. Whereas in Rhapsody there is flexibility to support synchronous and asynchronous signals which gives a greater depth of flexibility in interface design. Ease of use and flexibility is also very different across the four tools. With RoseRT, a capsule, a concept coming from ROOM methodology, must be used when defining Port interfaces. In both TauG2 and RoseRT Ports must be used and the interfaces defined up front in order to communicate between two "active" objects (objects which have their own thread of control). Rhapsody provides added flexibility in that designers can use Rapid ports to quickly create the port connections between components without having to define the interface details up front. Furthermore within Rhapsody, Ports do not need to be used between active or passive objects, which make it more practical in applications where Ports may prove to be too much of an overhead for simple communication patterns. Ports are great for encapsulation but they do present a level of overhead. Artisan's Real-time Studio does not contain UML 2.0 ports. It contains a graphical notation called a Port used to model ports on hardware components, but these do not contain any UML semantics.
- **Sequence Diagrams** – RoseRT and Real-time Studio are two MDD tools that do not support these enhancements. Telelogic TauG2 only supports the reference interaction occurrence that allows one to refer to other sequences within a sequence to create a more complex set of interactions and also allows a level of reuse of common scenarios. Rhapsody supports this as well as lifeline decomposition and easily links parent lifelines to the internal parts (internal lifelines of the parent) which makes it easy to navigate through the model top down to understand its decomposition. Artisan's Real-time Studio has a series of proprietary extensions to sequence diagrams that are not part of UML 2.0. They somewhat attempt to resemble a subset of UML 2.0 operators, but do not derive from the standard, nor are representative of the true semantics of UML 2.0 operators.

- **Information Flows** –Rhapsody is the only MDD tool that supports UML Information Flows. Both Tau G2 and RoseRT have no support of this concept. Artisan's Real-time Studio does have the concept of showing the flow of information between architectural components. However the flows are not compliant with UML 2.0 specifications, but rather non-standard extensions built into the tool. Real-time Studio does not let you describe the richness of flow items so that they can easily be realized in software. It is more of a documentation feature in the tool with no capability to describe the semantics of the flows. Rhapsody also uses this concept in a structural modeling technique, block modeling, which is supported by adding the concept of a block which extends the UML 2.0 structured class (see section on extending UML).

- **Structured Classes** – Three tools support the concept of Structured Classes and Real-time Studio supports the concept of decomposing system elements, but the elements again have no UML 2.0 semantics nor can they execute or show port interfaces to illustrate part connections. There was found to be a clear distinction between Rhapsody and TauG2 and RoseRT in their support of Structured Classes. RoseRT relates their capsule with that of a structured class and restricts one to having to use ports to connect the capsule to its internal parts through UML Connectors. This is the same with TauG2 in that one cannot directly connect parts to the structured class; they have to go through a port. While this is certainly a good design approach for some applications, in many it is not, as it requires overhead for a message to go through a port and in applications where quick response times are a must this is impractical. In Rhapsody, Ports are not forced on the designer and are used purely on an as needed basis, a direct connection between parts and the structured class can be easily done for rapid response.

- **State Machine Inheritance** – Both Rhapsody and TauG2 support State Machine Inheritance. Rhapsody's support of this new concept goes beyond TauG2 because TauG2 doesn't support some of the standard UML 1.X Statechart notations such as concurrency. In Rhapsody concurrency really presents an opportunity for capturing complex system behaviors when used in conjunction with State Machine Inheritance, in that states can be reused and then separated into concurrent states, which really gives the overall solution a great degree of flexibility. RoseRT supports State Machine Inheritance only for the capsules and also doesn't support many of the UML 1.X Statechart notations such as concurrency. RoseRT state machines are based on an alternative non-standard methodology called ROOM, which RoseRT has supported since its inception.

- **Activity Diagram** – RoseRT, Real-time Studio and Rhapsody support the notations of Activity Diagrams quite well. However RoseRT and Real-time Studio do not use this diagram for validation because it is a non-executable diagram within the tool. This diagram is mainly used to describe algorithm behavior and is heavily used within Avionics, Mil/Aero and systems engineering intensive systems, but in RoseRT it is limited to only analysis. In Rhapsody this diagram is executable so not only can it be used for analysis, it can also be used to validate the correctness of the system. TauG2 has no support for Activity Diagrams.

Model-Driven Development Productivity Enhancers

UML 2.0 is a standard set of graphical notations that enables Model-Driven Development (MDD). 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 Table lists the primary MDD technologies and their benefit to your organization and how the four primary MDD environments in the market support them.

Production Code Generation

Benefit – Get to final product quicker, and enables your designers to work at a higher level of abstraction in order to quickly create more complex systems. A lot of systems today have become so complex that handwriting the entire system would require years as opposed to months using this technology.

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

Quality is also crucial in safety critical systems where certification of the system is a requirement such as DO-178B.

TauG2	Rhapsody	RoseRT	Real-time Studio
Claim 80-90% generation Code proved to be unreadable, filled with macros and not structured in a manner that could be followed. Debugging the code is extremely difficult, not intuitive. For example it is difficult to know what pieces of the systems you are stepping through Certification is unlikely due to poor quality. Supports C with a subset of C++ constructs, no behavioral (Statecharts) code generation for C.	Claim 80%-90% generation Code is very readable, looks equivalent to handwritten. Their dynamic model/code view makes it intuitive to see how graphics map to code directly, so it hides nothing. Rhapsody promotes source level debugging with a variety of off the shelf IDEs in conjunction with their model execution Certification of the code is straightforward as they also promote customization and a rules based code generator. Rhapsody also targets certification because its execution framework is certifiable as is the code produced from its code generator. Supports full code generation for C, C++, Ada and Java.	Claim 80%-90% generation Code is readable, but it is not intuitive how the code maps to the model without a detailed understanding of the product. There is no dynamic mode/code view to allow one to directly see how graphics map to the generated code. RoseRT also promotes source level debugging with a variety of off the shelf IDEs in conjunction with model execution Certification seems possible, however RoseRT does not support a rules based code generation scheme, nor does it claimed to have a certifiable framework or code generator. Supports full code generation for C, C++ and Java	No real time framework - so doesn't claim any percentage of code generation. Real-time Studio generates only code frames that need to be manually tied to the RTOS, unlike the other three MDD environments. The behavioral code that is generated is for simulation purposes only, not deployable for production use.

Model/Code Associativity (Roundtrip engineering)

Benefit - Code and model always stay synchronized so that the graphical representation always represents the end production code, to ensure the code you are executing on your production system, is representative of your design and requirements.

This can be ensured by changing either code or model view and having the changes in one view synchronize with the other. The ease of which this is done gives your engineers a lot of flexibility in their design capabilities.

TauG2	Rhapsody	RoseRT	Real-time Studio
TauG2 does not support this concept. It is very model based and stresses that if there are defects found in the code, the defects should be fixed within the model not the code. From a software engineering perspective this is often unrealistic as many debugging sessions occur in the lab without the MDD even in the loop. You also need to figure out what to change in the model that reflects the code, which unfortunately is very hard to read.	Rhapsody supports this concept in many ways. It supports it dynamically, meaning that as soon as you make a change to the code, it will automatically reflect back into the model. This enables changes to be done on the fly. Rhapsody is the only tool to support dynamic model code view, which works with model/code synchronization so that as you click on the graphical design you will see the equivalent code in the view, which you can modify. Rhapsody will also pick up changes if the files have been modified outside of Rhapsody. It will detect the change once Rhapsody is loaded and will ask the user if they want to roundtrip the changes.	RoseRT also supports this concept, but with some restrictions. RoseRT will allow you to modify code however you need to go through a small process in order to synchronize the code. There is no dynamic on the fly synchronization. RoseRT also allows you to change files outside of the tool and upon bringing the tool back up it can detect the changes and ask the user if they would like to synchronize the code and the model.	Real-time Studio supports this concept However; the solution requires several steps in order to synchronize, such as bringing up an add-in tool to view the differences. Once the tool is invoked it is pretty straightforward to accept or reject changes made to the code. While this process is a bit cumbersome and certainly takes time, it is effective in ultimately having one consistent view of model and code

Reverse Engineering

Benefit - Allows the reuse of your IP, saving considerable time when new designs are begun.

The ability to graphically represent your legacy system in a variety of languages makes MDD more of a reality for those organizations where it would be too costly to try and start designing a system from scratch.

It is also crucial that an MDD solution produce an equivalent system during the code generation process, after reverse engineering. The reason of course is you want to ensure you IP functions the same when in the MDD environment.

TauG2	Rhapsody	RoseRT	Real-time Studio
TauG2 does have reverse engineering, but since the model in TauG2 is represented by an action language and not the actual code which was reverse engineered, it is difficult to tell how the model that was reverse engineered maps to your legacy system. This also ensures that the forward engineering process will produce code that is not at all representative of your IP, which makes reverse engineering with TauG2 of questionable value.	Rhapsody has reverse engineering modules for C, C++, Java and Ada. Rhapsody does bring in the source code into Rhapsody as a model and does bring in the function bodies exactly as they are in the legacy model. This is possible in Rhapsody since you don't need to represent the model with an action language; you do it directly with code. This means that the forward generation process will produce equivalent results to your IP, which makes reverse engineering of your IP a very viable solution with Rhapsody.	RoseRT does not have or claim to have a Reverse Engineering module with any of its implementation languages.	Real-time Studio has reverse engineering for C, C++, Java and Ada. Real-time Studio does also bring in the source code into the model as it was written and you can view the code within the model. Forward generation of the model would generate the same function bodies; the only drawback is the RE process is quite slow and doesn't pick up a lot of code dependencies, making it very difficult to actually build the model after reverse engineering without adding the dependencies manually.

Model Execution

Benefit - Allows you to validate the model at any point in development.

Constant validation of the system is key to producing high quality systems very quickly in that you are not waiting until all the design and implementation is done before you start testing the model. The closer that the model execution resembles the design execution on the target system, the greater the value provided by MDD.

TauG2	Rhapsody	RoseRT	Real-time Studio
TauG2 supports model execution, but uses a set of simulation services to exercise the model. While this may be good for very high level systems type analysis, it does little for software validation of the system, because the software system must ultimately run on the real time operating system of the target without the simulated services. TauG2 also seemed to translate the behavioral models that were done in C++ to C for simulation purposes, which indicates you cannot execute C++ behavioral models. TauG2 can simulate sequence diagrams and Statecharts.	Rhapsody supports model execution by using the real time operating system services at all times. Rhapsody does this through its real time framework. The application is portable enough, that you can simply take the same application that is running on a Windows or Solaris host and build onto a real time OS such as Integrity or VxWorks. Its abstraction layer makes the application independent of the services being used, so it is always using the services of the operating system to perform its execution. This has a lot of advantage because it mimics how the application is going to behave whether the application is running on the target or host. In fact the model can be executed directly on the target environment while simulation status is directed back to host environment so you can see in real time how the code is executing on the target. Rhapsody can simulate sequence diagrams, Statecharts and activity diagrams (it is the only tool that could simulate activity diagrams.	RoseRT supports model execution by using a set of virtual machine services in conjunction with some RTOS services. The RoseRT active object concept seems to use a set of simulation services for model execution of the active objects not the real operating system itself. RoseRT lets you run the virtual machine on the target system as well, but in many systems running the virtual machine on the target instead of directly using the RTOS services presents overhead and another layer of complexity. RoseRT can simulate on the target system as well with real time operating systems such as Integrity and VxWorks, along with their virtual machine services. RoseRT can simulate sequence diagrams and Statecharts.	Real-time Studio does not execute the model meant to run on the target system and what it can execute is very limited. It generates code just for simulation of the Statechart behavior of classes in the system. To build this simulation infrastructure is a three step process; generation of the state machine model, followed by the generation of the code and then the creation of a test harness which must be done in Microsoft Visual Studio. This type of simulation does not play well with an MDD environment where things must be executed continuously and often and the same code which is executing the model is the code which will be deployed on the target system. The other simulation capability Real-time Studio has is with an object animator, which really doesn't simulate the model at all, so this was not viewed as an MDD technology.

Conclusion

Embedded designs 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 higher quality. UML 2.0 is also being used as a starting point to allow systems engineers and software developers to work in one tool environment and thus increasing communication between the design teams. There are four major tool vendors that claim to support both the capabilities of MDD and UML 2.0. Each tool brings certain benefits to the user but based on analysis of the tools it is clear that the Rhapsody product from I-Logix provides the most advanced coverage of these capabilities, while extending itself beyond UML 2.0 to bring its benefits in a way natural to work habits of the systems designers and software developers.

Trademarks

APPENDIX

Model Driven Development Environment Buyer's Checklist

Key Criteria for Vendors when looking at a UML based Model-Driven Development Environment

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

	Criteria: Works with Legacy Systems
	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 XMI (or other interface)
	Criteria: Model Execution
	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)
	Complete behavioral code generation from models for the target environment (not code frames)
	Criteria: Production Code Generation
	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: Code Generation Languages:
	C++
	C
	Ada
	Java

	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:
	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
	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 shelve integrations with 3rd 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

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