Model-Driven Development for Embedded Systems

A case study in Rhapsody
Deployment on Xilinx FPGA platforms

By Senior Consultant Kim Bjerne (kim.bjerne@teknologisk.dk)
Copyright © 2008 Danish Technological Institute

Preface ............................................................................................................................................... 2
UML 2.1 diagrams ................................................................................................................................... 3
UML usage levels ..................................................................................................................................... 4
Model-Driven Development .................................................................................................................... 5
Rhapsody a Model-Driven Development tool ....................................................................................... 7
Rhapsody Frameworks ............................................................................................................................. 10
Design Pattern Example .......................................................................................................................... 11
Sorter Induction Example ........................................................................................................................ 14
Deployment on an embedded FPGA platform .......................................................................................... 19
Linux PowerPC platform ......................................................................................................................... 21
Xilinx Kernel Microblaze platform ........................................................................................................... 21
Microblaze platform without RTOS .......................................................................................................... 21
Creating platform configurations ............................................................................................................. 22
Debugging and testing ............................................................................................................................... 26
Summary ................................................................................................................................................ 28
References ............................................................................................................................................ 29
Preface

The Unified Modeling Language (UML) is a general purpose visual modeling language for system development. The audience for this article is software developers and system architects who wish to obtain an overview of how to use UML for Model-Driven Development (MDD) for real-time and embedded systems.

The principle of MDD is a paradigm which promotes the use of models at different levels of abstraction and transformation between them in order to drive a concrete application implementation. MDD promotes the construction of models at a high level which can then be transformed to the final implementation platform.

Our approach is Model-Driven Development based on UML which gives the developer a high level of abstraction and where focus is moved from coding, language and platform details to specification and design. It allows an iterative development and verification process on a higher abstraction level than pure coding.

It is important to carry out transformation so that there is complete confidence between the views of the model. This approach calls for automated transformation to reduce development time and cost.

The benefits will be a faster, improved and robust design which is platform-independent and easily moved to different hardware and software architectural platforms. It allows you to start software development before hardware is available or even to move the model when the hardware platform is changed.

The last part of this document contains different model examples based on UML implemented in Rhapsody where the models have been deployed on different HW/SW architectures. In this case study the embedded platform is a Xilinx FPGA running an embedded 32bit microprocessor (Microblaze or PowerPC).

The configuration and deployment of the models is performed without an operating system, with the real-time operating system (Xilinx kernel) or embedded Linux. The examples demonstrate that it is possible to separate the model-driven application development from the platform implementation.
UML 2.1 diagrams

UML helps you specify, visualize and document models of embedded software systems, including their structure and design, in a way that meets the system requirements. You can model just about any type of application, implemented on any type and combination of hardware, operating system, programming language or network.

UML 2.0 defines a number of diagram types that can be divided into three categories. Structure diagrams include: class, structure, object, package, components and deployment diagram. These are all basically different, not with regard to the contents of the diagram but with regard to purpose. Behaviour diagrams focus on the specification of behaviour of individual elements covering state machine and activity diagrams. Use-case diagrams are used to emphasize functionality, identification of system interface (actors) and use-case scenarios. They focus on the specification of system usages. Interaction diagrams focus on how elements (objects) collaborate together over time to achieve functional goals.

Structure Diagrams

- Class and Object diagram
- Composite Structure diagram
- Package diagram
- Component diagram
- Deployment diagram

Behavior Diagrams

- Use-case diagram
- Activity diagram
- State Machine diagram

Interaction Diagrams

- Sequence diagram
- Timing diagram
- Communication diagram
UML usage levels

The use of UML diagrams for embedded system development can be divided into three levels depending on the ambition of usage and integration of automated code generation.

Informal level
The UML diagrams are used in the analysis and design phases of the project to sketch a few diagrams from where the coding proceeds. In this approach the diagrams are used to get a common understanding of the system to be constructed. UML is only used in the analysis and design phases of the project and is not likely to be updated during the implementation phase.

Structural level
The UML diagrams are used together with a tool to generate codes and the developer fills in the rest by using the model as a guide. This approach saves the programmer from manually creating all the classes and relations identified in the design model. A tool like Enterprise Architect\(^1\) is able to support this development process. It is also possible to import existing code into the UML model. This could be used for reengineering and documentation of existing systems.

Executable level
This is the Model-Driven Development approach. The model views in this approach and the generated source code has a close relation. Analysis, design, implementation and tests are all integrated into the UML tool. Very early in the design it is possible to actually execute and verify the model-based on structural and behavioural UML diagrams. This kind of development changes the focus from the art of programming to application development and abstracts the underlying hardware and operating system. A tool like Rhapsody from Telelogic supports this approach.

---

\(^1\) Enterprise Architect from Sparx Systems, supports UML 2.1 diagrams and code generation
Model-Driven Development

The purpose of this chapter is to introduce Model-Driven Development using UML at the “Executable level”.

The principle of Model-Driven Development (MDD) is a paradigm which promotes the use of models at different levels of abstraction and transformation between them to drive a concrete application implementation. A model is expressed in a language at some abstraction level. Models are used to increase productivity, and MDD offers the potential for automatic transformation of models into other models. The source and target languages of the transformation can be defined by metamodels. A metamodel is yet another abstraction highlighting the properties of a model. UML is described by terms of a metamodel.

The use of UML for model-driven software development gives a high level of abstraction and a visual view on the design. The abstraction will be moved from coding and language details to focus on the specification and design of the software system to be implemented. The model-driven approach tries to decouple the design from realization. It allows the developer to focus on what the system should do instead of trying to cover all details in one step.

The UML design model is translated into an executable specification which can be tested and visually verified very early in the design process. Model-Driven Architecture (MDA) defines an approach to software development that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform.

The first step is to create a Platform-Independent Model (PIM) of the structure and functions of the system to be modelled. The second step is to map this functionality to the Platform Definition Model (PDM). The Platform-Specific Model (PSM) specifies the realization of the functionality in PIM on a specific platform. In this case it could be an embedded PowerPC platform running embedded Linux.

The model-driven development process relies on a number of important principles:

- **Use of Models** – A large complex system is very difficult to construct effectively by only using source-code-level construction. Models provide a way of thinking of how to represent domain concepts as model elements. These models permit the developers to capture important characteristics of the application and they interrelate. The use of UML diagrams allows the designer to create different views of the model.

- **Iterative Development** – Is based on incremental construction where it is possible to start with a part of the system. It is able to make rapid prototypes with production-quality software parts which can be tested and released.

- **Executable Models** – It is only possible to test things which execute. Therefore, to be able to perform test-based development, it is required to execute the model early and often in the design process. The key is model-based translation of designs so that the transformation of a design into something which executes only takes a short time.
• **Model Code Bidirectional Associativity** - For model-based systems, it is absolutely crucial that the code and diagrams are different views of the very same underlying model. Therefore changes in the code must be able to reflect UML views, and changes in UML views must be updated in the code. Separate maintenance of code and model becomes very difficult, and the result is normally that the model is only used in the start of the development process.

Some of the benefits for using Model-Driven Development are listed below:

• **To deal with complexity of system development**
  o Abstract a problem to focus on some particular points of interest
  o Improve understandability of a problem
  o Possible set of nearly independent views of a model
  o Iterative modeling may be expressed at different levels of abstraction

• **To minimize development risks**
  o Through analysis and experimentation performed early in the design cycle
  o Enable to investigate and compare alternative solutions

• **To improve communication - information sharing and reuse**
  o A good model is better than a long speech or legacy source code

• **To reduce development mistakes**
  o Automatic model transformation is less subject to error than the writing of your own code
Rhapsody a Model-Driven Development tool

The purpose of this chapter is to present a brief overview of Telelogic Rhapsody in C++ developer edition.

Rhapsody in C++ is targeting embedded software development and makes it possible to create a platform-independent model. It covers automated code generation in C++ including configuration of environment and makefile generation. This approach makes it possible to set up a cross compiler and linker to be used for the target platform.

Visual models can be executed and animated for early verification and test. Rhapsody supports development on Windows and Linux. When the model has been deployed to the target platform, it is possible to debug the code with integration of Rhapsody for model animation.

Rhapsody covers all the elements required for Model-Driven Development based on UML with the following basis functionalities:

- Visual Modeling with UML
- Model Execution and animation
- Platform-independent (PIM)
- Platform-specific (PSM) translation of PIM to technology platform
- Real-Time Framework (Abstracts the underlying OS)
- Language support for Java, C++ and C
- Model-Code Bidirectional Associativity
- Rule-Based Code Generation
- Automated Requirement-Based Testing
- Reverse Engineering

Figure 1 Key concepts of Rhapsody
Rhapsody includes an IDE which allows you to develop your system where code and model can be viewed and modified in the same environment. It is possible to execute and visualize your model by animation of dynamic behaviour for state, sequence and activity diagrams.

Figure 2 Rhapsody Integrated Development Environment (IDE)
Below is listed my experience working with Model-Driven Development using Rhapsody compared to traditional software development.

**Pros:**
- UML diagrams are integrated into the development process
- Visual overview of design and architecture
- Auto-code generation of the UML model
- Good integration for design and code
- Unified implementation
- Fast implementation
- Continuous simulation and verification
- Good support for an iterative and agile development process

**Cons:**
- Tool costs are high
- Demands on effort and time for education
- New way of working – it takes time
- Few educated and experienced MDD developers
- Less optimal code than handwriting in some cases
- How to handle big chunks of legacy code? (Reverse Engineering)
- Difficult to make verification of timing requirements

It is recommended for reuse of old legacy code to wrap it into libraries linked with the model, and then only import the header files into the Rhapsody model. (Import As External)

It is possible to configure how the Rhapsody code is generated by changing the properties for project, package or class. In some situations, this could be required to optimize the code size.

Many properties in Rhapsody can be modified and changed at different project levels. The properties make the tool very flexible but it also makes it difficult to maintain the model afterwards. A written policy for how and when to change properties could alleviate this problem.
Rhapsody Frameworks

Rhapsody generates code by using a framework which abstracts the underlying operating system. Two basic frameworks are used in the following case to demonstrate how models can be moved from one platform architecture to another without changing the model. From the model developer’s point of view, the task would be to set up Rhapsody generating the source code, compile and link it to the embedded platform.

The Object eXecution Framework (OXF) is the framework abstracting the operating system from the model. Rhapsody includes OXF adapters to support a number of common used RTOS for embedded systems like: VxWorks, Integrity, Nucleus and Linux. OXF consists of three main parts:

- Event-Driven Framework (Core API and Implementation package)
  - Thread, Reactive, Event, Timeout, Protected, Time and Memory Manager
- Operation System Adapter Layer
  - (Adapters package – must be changed for the specific OS)
- Container Classes (Part of services package)

![Figure 3 Rhapsody OXF Framework with RTOS](image)

The Interrupt Drive Framework (IDF) makes it possible to use Rhapsody in C++ without an operation system. The IDF replaces the OXF and RTOS and generates smaller code size. It only requires a periodic timer interrupt which must be triggered from a hardware timer interrupt. The IDF framework is not part of the Rhapsody distribution but is available upon request from Telelogic.

![Figure 4 Rhapsody IDF Framework without RTOS](image)
Design Pattern Example

A simple model has been created in Rhapsody which is based on a course exercise in “OO Design Patterns”. This example covers a number of software design patterns including visitor, composite, factory, strategy and mediator. The model is described in a class, sequence and state diagram using Rhapsody. See figure below.

Figure 5 Class diagram of the Visitor Design Pattern

The class diagram describes the structure, including classes and their relations. Methods and attributes are added to the class diagram and the required functionality implemented in methods using C++ code.

Figure 6 Code entered in the CalculateMTBF method of the Client class

The sequence and state diagrams describe the dynamic behaviour of the model.
Figure 7 Sequence diagram of the Visitor Design Pattern

Figure 8 State diagram for the Visitor Tester class
Finally the model has been tested in Rhapsody by animating the model execution. Verification of result is performed by animating the sequence and state diagrams of the model. The expected output is tested by printing to the output console.

---

Figure 9 Animated Sequence diagram of the Visitor Design Pattern

Figure 10 Animated State diagram of the tester behaviour
Sorter Induction Example

This is a model of a controller which handles induction of parcels on a sortation machine. The induction consists of four belts, each serving different functions for tracking, measuring, coding and inducting parcels onto the sortation machine. See Figure 11 The “sorter induction” model.

A bar-code on the parcel is scanned by the operator and the length of the parcel is measured before the parcel is inducted on the sorter machine. The induction has a connection to a Central Sorter Controller (CSC) which serves as the central computer for sortation of parcels for the whole sortation system. The CSC assigns trays on the sorter for the parcels to be inducted and the Card Reader searches for the tray assigned by the CSC.

Figure 11 The “sorter induction” model

The “sorter induction” model contains simulation of parcels being inducted. The induction simulator emulates the environment of induction. The purpose is to test the induction model without hardware being available. The design is set up so different configurations can be created with and without the simulator.

The simulator contains a parcel generator and a photo sensor simulator which are connected as components. It simulates photo sensors (PEC) being dimmed. It simulates parcels on its way,
passing belt and PEC’s. The simulator contains a key board interface which handles commands from a console to operate the induction simulator.

The following development steps have been performed to create and test the “sorter induction” model in Rhapsody and deployment of the model on the target platform.

- **Analysis for domain and application**
  - Use-Case analysis
  - Class and object diagrams
  - Scenarios

- **Design and Architecture**
  - Subsystem design
    - Structure views
  - Packages
    - Layers and dependencies
  - Refined class diagrams
    - Process and concurrency
  - Use-Case scenarios
    - Sequence diagrams
  - Object behaviour
    - State machines

- **Simulation and automated test generation**
  - Environment simulation
  - Test generation

- **Animation and verification**
  - Sequence diagrams
  - State machine diagrams

- **Deployment and timing on target**
  - Xilinx FPGA board running Microblaze soft-core processor
  - Rhapsody framework and configuration
  - Eclipse IDE to build and debug
  - Hardware buttons and LEDs for testing
  - Serial console for testing and time verification

The following figures illustrate usages of different UML diagrams covering: structure, behaviour and interaction diagrams created in the modeling of the induction.
Figure 12 Class diagram for the Belt Controller of the “sorter induction” model

Figure 13 Composite Structure diagram of the “sorter induction” model
Figure 14 State machine diagram for the measurement belt controller class

Figure 15 Animated sequence diagram for the use-case scenario “parcel measurement”
One of the important areas for real-time embedded systems is timing requirements. For the “sorter induction” model it is important that parcels are measured and detected and belts are controlled with a precision of less than 20 ms. With a belt speed of 1 m/s, a parcel will move 1 cm in 10 ms. Verification of timing is performed in terms of debug messages which are logged with timestamp in ms. The output log of model execution is shown in Figure 16 Test log for running the “sorter induction” model.

![Figure 16 Test log for running the “sorter induction” model](image-url)
Deployment on an embedded FPGA platform

In addition, the “visitor” and “sorter induction” model described in the previous chapters, a “stop watch” model has been created to try with different model mappings to different target platforms.

The “stop watch” model uses button inputs on the embedded FPGA platform to start, stop and reset the watch. The purpose is to demonstrate how it is possible to use external hardware components in the model deployed on the embedded platform.

Figure 17 Composite Structure diagram of the “stop watch” model
The first target platform in this case is a ML405 evaluation board from Xilinx which has a Virtex 4 FPGA with an embedded PowerPC running Linux 2.6.

![Xilinx FPGA ML405 evaluation board](image18)

**Figure 18** Xilinx FPGA ML405 evaluation board

The second platform tested is the Spartan 3E starter kit running a Microblaze 32bit soft-core processor running with the Xilinx kernel or without any RTOS.

![Xilinx FPGA Spartan 3E starter kit](image19)

**Figure 19** Xilinx FPGA Spartan 3E starter kit

The challenge is to configure Rhapsody to generated source code for the different combinations of architectures (processor and RTOS).
**Linux PowerPC platform**

The OXF framework has been modified to support a Linux 2.6 executed on a PowerPC.

![Figure 20 Rhapsody OXF Framework with Linux and PowerPC](image)

**Xilinx Kernel Microblaze platform**

The OXF framework has been modified to support a RTOS-Xilinx kernel executed on a Microblaze soft-core processor.

![Figure 21 Rhapsody OXF Framework with XilKernel and Microblaze](image)

**Microblaze platform without RTOS**

The IDF framework has been modified to be driven from a hardware timer interrupt generated from the Microblaze soft-core processor.

![Figure 22 Rhapsody IDF Framework without RTOS](image)
Creating platform configurations

Rhapsody has many properties which allow customization on the tool and the generated code. Properties can be set once and for all by modifying the site.prp file in the “share\properties” directory. A property hierarchy allows you to change properties on different levels. In our case we will change the platform settings in the “siteC++\ppr” file.

![Rhapsody properties directory](image.png)
Adding a new platform is performed by creating a new configuration for the property 
**Environment in “siteC++.prp”**. Description of new environments are made in the included 
“*.prp” files. These files contain the description for the platform tool chain including: Compiler 
settings, linker, include files and makefile content.

**Contents of “C:\Programmer\Telelogic\Rhapsody 7.2\Share\Properties\siteC++.prp”:**

```
Include "CPP_RulesComposer.prp"
Include "siteC++_IDF_Gnu_Microblaze.prp"
Include "siteC++_OXF_Gnu_MBXilkernel.prp"
Include "siteC++_OXF_Gnu_PPCXilkernel.prp"
Include "siteC++_IDF_Microsoft_NT.prp"
```

In addition to the modification of the property files, new source code should be added to support 
the Xilinx kernel for the OXF adapter layer. This is done by creating the MBXilkernelos files and 
adding makefiles in “C:\Programmer\Telelogic\Rhapsody 7.2\Share\LangCpp\oxf” directory. See 
figure below.

![Rhapsody OXF framework with MBXilkernel support](image-url)
The last modification is to integrate the IDP framework into the Rhapsody model by adding the IDPProfile. The IDP framework is written as a Rhapsody model itself where some basis adapter functions has to be modified to support the Microblaze processor.

Figure 25 Rhapsody IDP framework

IDF functions to be modified cover: OXFGetSystemTick, OXFInitTimer, OXFSleep, OXFTick, OXFMaskInterrupts, OXFUnmaskInterrupts.
When the configuration of Rhapsody is completed, the only task for the model developer is to create a new configuration and set the environment for where to deploy the model. In our case study it would be one of the following:

- IDF_Microblaze
- MBXilkernel
- PPCLinux

The figure below shows how to set up a new Environment for the Microblaze Xilinx Kernel. The compiler options are automatically set for the Microblaze GNU compiler based on the property settings created in the “siteC++.prp” file.

![Figure 26 Creating platform configuration in components](image)

![Figure 27 Create a new configuration in Rhapsody and select environment](image)
Debugging and testing

For the ML405 target platform running Linux on a PowerPC, the Rhapsody models are developed and tested on Linux and deployed to a PowerPC Linux architecture. As the Rhapsody OXF already supports Linux, it is only a question of setting up Rhapsody to use the PowerPC GNU cross compiler. This is the simple case where everything is performed within Rhapsody. The binary output file is copied to the target platform where the result is verified and tested on the target.

For the Spartan 3E starter kit running a Microblaze soft-core processor, the Rhapsody models are developed and tested on Windows. The models are deployed to the Microblaze architecture with the Xilinx kernel and without kernel. The generated source code is added to a project in the Xilinx Platform Studio – Software Development Kit (XPS-SDK). XPS-SDK is an eclipse based IDE with compiler and debug integration for the Microblaze soft-core processor.

In the “stop watch” model, source code has been added to support the hardware interrupt functions to generate a triggerInterrupt to the IDF framework for button operations.
The model is finally verified by using the Hyper Terminal console connected to the serial port of the Spartan 3E FPGA target platform.

```
DEBUG: 36696 InductionBelt: P1 : PecDim : C12300010 : L0.80 : T 5
DEBUG: 36704 InductionBelt: P1 : Stop : C12300010 : L0.80 : T 5
DEBUG: 36964 InductionBelt: P1 : CardDetect : C12300010 : L0.80 : T 5
DEBUG: 3778 MeasureBelt: P2 : Length : C12300020 : L0.79 : T 0
DEBUG: 38684 InductionBelt: P1 : Inducted : C12300010 : L0.80 : T 5
DEBUG: 38964 TransitionBelt: P1 : Transfer : C12300010 : L0.80 : T 5
DEBUG: 38730 MeasureBelt: P2 : PecDim : C12300020 : L0.79 : T 0
DEBUG: 38740 MeasureBelt: P2 : Start : C12300020 : L0.79 : T 0
DEBUG: 38748 MeasureBelt: P2 : LeftBelt : C12300020 : L0.79 : T 0
DEBUG: 40686 MeasureBelt: P2 : Empty : C12300020 : L0.79 : T 0
DEBUG: 40782 InductionBelt: P2 : CardAssign : C12300020 : L0.79 : T 10
DEBUG: 41228 InductionBelt: P2 : PecDim : C12300020 : L0.79 : T 10
DEBUG: 41236 InductionBelt: P2 : Stop : C12300020 : L0.79 : T 10
DEBUG: 41498 InductionBelt: P2 : CardDetect : C12300020 : L0.79 : T 10
DEBUG: 42000 InductionBelt: P2 : Inducted : C12300020 : L0.79 : T 10
DEBUG: 42910 TransitionBelt: P2 : Transfer : C12300020 : L0.79 : T 10
```

**Figure 29** XPS-EDK Eclipse IDE integrated with the XMD debugger

**Figure 30** The “sorter induction” model running on the Spartan 3E FPGA platform with RTOS
Summary

This article has demonstrated how the use of Model-Driven Development (MDD) based on UML can be applied for embedded software development. The Rhapsody tool has been applied to show how visual UML models can be deployed to different target platforms. A case study illustrates how it is possible to develop UML models and to deploy them on different Xilinx FPGA platforms, including embedded processors (Microblaze and PowerPC) running different operating systems (Xilinx Kernel, Linux and without OS).

Separation of the model from the platform allows development of software on a higher abstraction level and allows the developer to verify the model by visual animation and execution on the development platform (Windows or Linux).

Rhapsody provides you with a development environment where it is possible to use the UML diagrams in all development phases of a project, all the way from use-case analyses to design and implementation. The code is auto-generated from the UML model, and there is a fine integration between model and code. It provides a development environment where the UML diagrams can be used as part of the coding and implementation process. This approach opens a new approach of working. It takes time to learn but it opens a new world where benefits in terms of increased productivity, portability and higher quality of design and code are worth considering.

Education is one of the major challenges for the MDD approach. It takes time to come from a traditional coding level to the use of UML diagrams and to become an expert using a MDD tool like Rhapsody. The next chapter shows a list of books and training course material which has formed the basis for this article and case study in Rhapsody.

You also have to decide on a methodology which specifies how to use the different UML diagrams in the development process. You can choose to define your own or to use some of the methodologies already described and in use in the industry. Below is shown some of the most common methodologies for embedded systems.

- The **Unified Process** (UP)
- **Teknologisk Unified Process** (TUP)
- The **Rapid Object-oriented Process for Embedded Systems** (ROPES)
- The **Harmony** Process (Newest revision of ROPES)

It is recommended to adapt the chosen methodology to own needs. See references, the last chapter for books describing these methodologies.
References
Below find books, training material and tools which have been used as background material for this article.

Books
The following books will serve as a good starting point for learning about UML, Model-Driven Development and design patterns for real-time systems.

“UML and the Unified Process, Practical Object-Oriented Analysis & Design” by Jin Arlow, Ila Neustadt (Describes the Unified Process)

“Real Time UML Third Edition” by Bruce Powel Douglass
(Describes the ROPES process)

“Real-Time UML Workshop for Embedded Systems” by Bruce Powel Douglass
(Describes the Harmony process)

“Real-Time Design Patterns” by Bruce Powel Douglas

Training
The following training material has been used as input for this article:

“OO analyse, design og realisering med UML”, course material, Danish Technological Institute (Describes the “Teknologisk Unified Process” TUP)

“OO Design Patterns”, course material, Danish Technological Institute

“Essential Tool Training Rhapsody C++”, course material Telelogic

Links and tools
OMG Group web site: http://www.uml.org/
Telelogic - Rhapsody (UML-MDD tool) web site: http://www.telelogic.com/
Xilinx – FPGA tools and boards web site: http://www.xilinx.com/

Tool versions
Telelogic Rhapsody in C++ ver. 7.2
Xilinx FPGA tools: EDK and ISE ver. 9.2
V60_RiCppInterruptDrivenFramework_rpy ver. 6.0