MODEL DRIVEN ARCHITECTURE, CONTROL SYSTEMS AND ECLIPSE

A. Vodovnik[1], K. Žagar[1]
[1]*Cosylab, Ljubljana, Slovenia*

ABSTRACT

In the modern world, the increasing complexity of control systems is required to meet up with demands for more and more complex instrumentation techniques. This kind of equipment often consists of many components that require controlling and monitoring. Not only that, but it also requires that the control system is distributed, dependable, fault-free and yet also maintainable, secure and manageable.

For the control system architects and developers this presents a formidable task alleviated in part by using a Model Driven Architecture (MDA) approach. This is a technique employed by software engineers to firstly model the essence of the control system and only then focus on further development. As opposed to non-MDA approaches, the model is not merely a form of documentation for the control system but rather a central artifact. By using tools such as program generators or Computer Aided Software Engineering (CASE) applications, other elements of the control system are generated – such as the source code, documentation and software references.

This article focuses on outlining the current evolution of such a development approach and the tools aiding it. It also focuses on the use of a wide-spread open-source IDE Eclipse [4] to combine the MDA approach with standard development tools.

Based on previous work by Cosylab in the fields of Control System Modeling Language (CSML) and XML program generation, a prototype plug-in solution for the Eclipse IDE is also presented in this article. The prototype presented is likely to become a part of the Control System Office Suite which we are currently developing.

INTRODUCTION

Today, one of the key standards for Model Driven Development is the *Model Driven Architecture* (MDA) [1] conceived by the Object Management Group's (OMG) vision of a model based development. The approach suggests that abstract models, which are platform independent, are transformed in a systematic way to generate deployable, platform-specific implementations.

These transformations offer a materialization of collections of design decisions an architect is required to make in order to satisfy requirements of the application being developed. The goal of this article is to show that the same approach can be applied when developing control systems and doing so by using tools either already present on the market (or in the open-source community) or tools that are not too complex to develop.

MODEL DRIVEN ARCHITECTURE

Introduction to MDA

In today's business, IT is said [9] to be best for enterprise when serving business first and technology second. However, this is hard to achieve when the most powerful tools feature exclusively on technology aspects. MDA bridges the gap between the business modeling and the interfaces used by enterprises all over the world.

The important question arising with this topic is why is modeling really important? For one, it allows the developed application, be it an enterprise solution for financial planning and cost modeling, distributed telecommunications signaling tracing or a control system for the most complex accelerators and other large experimental devices, to be easily maintained, yet scalable and secure. It also alleviates the programmer of the formidable task of having to develop the application with all the components of the system already in mind with no way to analyze them first. Models can represent the applications strengths and weaknesses before the application is even written – saving the developer's effort. They can represent an object (e.g. a device) at an abstract level and in a platform independent way.

The MDA requires that one definitive PIM, or Platform independent model, be created first when developing an application. From this model, using platform specific mappings, generators and predefined patterns and templates, a platform specific model (PSM) is generated – for example as general as Java, .J2EE and.NET or as specific as an EPICS Asyn driver as in the case of this article

Because MDA development focuses primarily on the functionality and behavior of distributed, complex applications, the architect, when designing the PIM, does not need to shift his attention to idiosyncrasies of the technology platform on which the application will be implemented. This in fact, is the task of the developer that has a very good understating and overall knowledge of the platform he is implementing with.

After the creation of models and study of the architecture, the software architect can make modifications related to any part of the application. If the application were already written and a crucial flaw would emerge (e.g. a performance issue in error handling) it could take a while to redefine the architecture and rewrite the impacted classes, in turn wasting effort of both the architect and the developer. Using the MDA however, the architect could simply modify the PIM and regenerate the application with no other effort from the developers.

Using the PSM, another generator transforms the model intro artifacts – the documentation, implementations, automatic tests (JUnit), design documents, API documentation …

UML, MOF and their role in the approach

UML is often thought of as being the crucial part of MDA. This is because of its visualization capabilities – all diagrams, abstract models, whether PIM or PSM, may be drawn with it. It is common to define models using this language. In reality however, the most important part of the MDA is the Meta Object Facility. It allows for UML structural and behavior models to be transmitted via XMI (XML Metadata Interchange) to any MOF-compliant repository from where they can be shared with others. According to OMG's specification of the MOF, its central theme is to provide extensibility. Because of its layered metadata model, it allows for new kinds to be easily added. Figure 1 shows an example of the four-layered model. The first layer, the meta-metamodel is the top most layer. It defines abstract items such as the MetaModel, MetaClass, MetaAttribute. The next level is the meta-model. This model serves as the basis for models defined at the model level. Note that, although in the example, the model of "Devices" is chosen, the meta-model could just as easily be used to describe, for example a library of articles, or a stock exchange. The last layer, the fourth layer or the information layer, is an actual representation of the abstract model defined in the third layer.

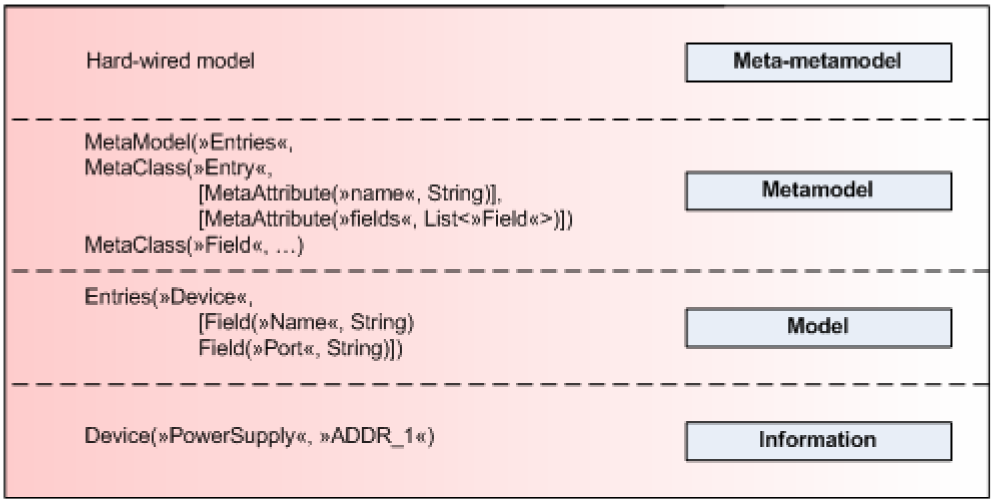


Figure 1 MOF multi layered presentation

Platform specific Model

As the platform specific model is generated from the PIM it simply reduces the abstractness of the PIM. It becomes more specific to the platform, to put it simple. Imagine a device with a property "name". To generate a PSM, one must decide to which platform it will be bound. If the choice is made to implement this in C#, the PSM would have to incorporate another private field (private string *fName* for example) and a public property Name.

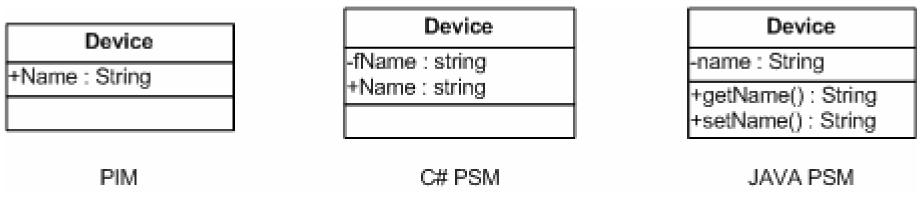


Figure 2 UML representation of the PIM and PSM models related to the example.

Platform specific implementation for C#	Platform specific implementation for Java

`private string fName;`
`public string Name {`
`      get   {`
`            return fName;`
`      }`
`      set {`
`            fName = value;`
`      }`
`}`
	`private String name;`
	`public String getName() {`
	`      return fName;`
	`}`
	
	`public void setName(String value) {`
	`      fName = value;`
	`}`

The examples show that, although the PIM and PSM are related, they are not the same. Because of this, keeping them synchronized is also a formidable task. Many companies opt for skipping either the PIM or the PSM. Standard practice however, shows that currently, the developers are focuses more on PSM while completely ignoring the PIM.

ECLIPSE

Introduction to Eclipse

Eclipse is an open source Java IDE [4] that is widely used. For example, at Cosylab we have been using Eclipse as our primary development tool since the year 2002. Because of its widespread adoption, more and more Eclipse plug-ins are being developed that allow the users to accomplish more tasks in one environment. Because Eclipse focuses on providing an extensible framework, one can already see hundreds of tools emerging that span the range from business intelligence, modeling, graphical editing and similar…

Support for MDA

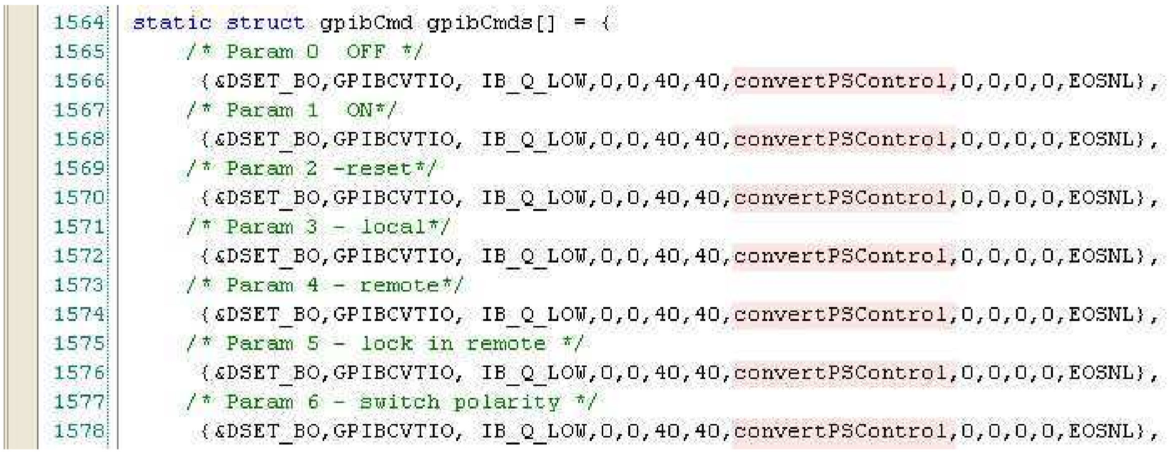
As more and more users realize the importance and the role MDA will play in software development in the future, more tools are beginning to emerge. For Eclipse, no open source, free tool is currently fully implemented that would support MDA as a whole but there are several projects emerging which are beginning to show that this might some day be achieved. These include The Eclipse Model Driven Development Integration tool [5] or the Eclipse Generative Model Transformer [6]. Both of these are still in an early stage of development but the latter already has some results available for download. There are however, some corporate solutions, for example the IBM Rational XDE that have a fairly good support for MDA.

Eclipse however, has quite a good support for UML 2 [7] based on the Eclipse Modeling Framework model [8]. This model will also serve as the basis for our prototype solution.

MDA SUPPORT FOR ASYN DRIVER

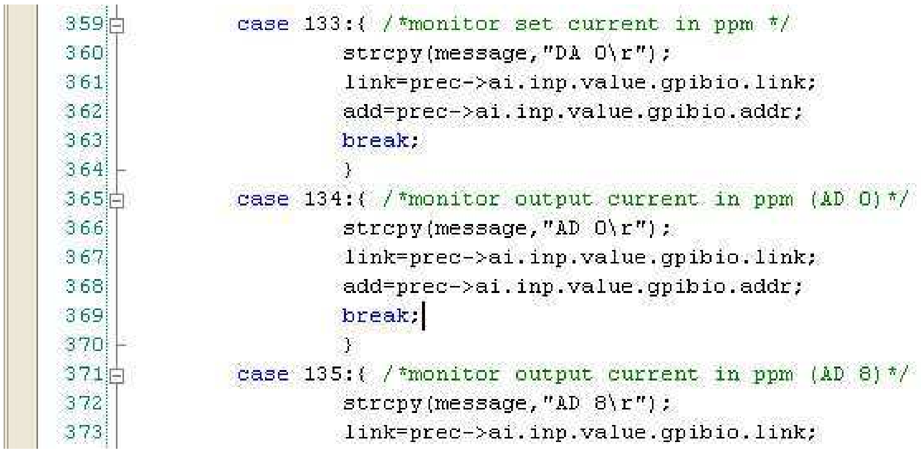
Introduction to the problem

One of the common problems with new devices is the lack of drivers for them. Implementing them requires extensive knowledge of the device, its specification and is a tedious task. It frequently involves writing hundreds of repetitive lines of code. This article focuses on illustrating the problem with an example of writing EPICS asyn drivers.



```
static struct gpibCmd gpibCmds[] = {
    /* Param 0  OFF */
     {&DSET_BO,GPIBCVTIO, IB_Q_LOW,0,0,40,40,convertPSControl,0,0,0,0,EOSNL},
    /* Param 1  ON*/
     {&DSET_BO,GPIBCVTIO, IB_Q_LOW,0,0,40,40,convertPSControl,0,0,0,0,EOSNL},
    /* Param 2 -reset*/
     {&DSET_BO,GPIBCVTIO, IB_Q_LOW,0,0,40,40,convertPSControl,0,0,0,0,EOSNL},
    /* Param 3 - local*/
     {&DSET_BO,GPIBCVTIO, IB_Q_LOW,0,0,40,40,convertPSControl,0,0,0,0,EOSNL},
    /* Param 4 - remote*/
     {&DSET_BO,GPIBCVTIO, IB_Q_LOW,0,0,40,40,convertPSControl,0,0,0,0,EOSNL},
    /* Param 5 - lock in remote */
     {&DSET_BO,GPIBCVTIO, IB_Q_LOW,0,0,40,40,convertPSControl,0,0,0,0,EOSNL},
    /* Param 6 - switch polarity */
     {&DSET_BO,GPIBCVTIO, IB_Q_LOW,0,0,40,40,convertPSControl,0,0,0,0,EOSNL},
```

Figure 3 Asyn driver source code; command definitions.



```
        case 133:{ /*monitor set current in ppm */
                strcpy(message,"DA 0\r");
                link=prec->ai.inp.value.gpibio.link;
                add=prec->ai.inp.value.gpibio.addr;
                break;
                }
        case 134:{ /*monitor output current in ppm (AD 0)*/
                strcpy(message,"AD 0\r");
                link=prec->ai.inp.value.gpibio.link;
                add=prec->ai.inp.value.gpibio.addr;
                break;
                }
        case 135:{ /*monitor output current in ppm (AD 8)*/
                strcpy(message,"AD 8\r");
                link=prec->ai.inp.value.gpibio.link;
```

Figure 4 Case sentences.

Figure 3 shows the sample source code for an asyn driver created for a power supply device. The code is almost the same in every line. However, it is linked to another batch of code, located by a pointer to a function convertPSControl. In this function a number of case sentences exist, which can be observed in Figure 4. For each of the above definitions of parameters in the struct, a case sentence must be present. Maintaining such code, with links based on indices is hard and time consuming.

Proposed solution

The proposed solution presented in this article is to model the device in an abstract manner. Through the help of the MDA Support for Asyn Driver the article shows that it is possible to derive an EPICS Asyn driver from a PIM model. Because the primary development tool at Cosylab is Eclipse, it was also a goal of the implementer to use Eclipse IDE and as much of the available tools and frameworks. A good tool for this is the EMF, the Eclipse Modeling Framework. The Eclipse Modeling Framework already offers three major packages crucial for this prototype:

1. **The Core EMF**
   This includes the core support for describing models and provides runtime abilities for change notification, persistence etc.
2. **The EMF.Edit package**
   This package provides generic, reusable tools for editing the description of models and a generic set of commands to ease the development of editors for the models described with the EMF.
3. **The EMF.Codegen package**

> This package contains all the facilities needed to generate the code for the editor of the EMF model. With this editor, the driver developers will model the device and, as an end result, see the driver and documentation “pop out”.

By introducing a well-planned model into the EMF framework, one can generate the required editors to allow developers of drivers to visually design the code. A part of the MDA Support for Asyn Driver is also an extension to the generated editor for including some support for code generation. The intention is to demonstrate that this type of tool is possible and test if further development is feasible.

The architecture

Because it is not the intention of this article to further complicate the development of drivers but rather to simplify them, the focus will be to keeping everything as simple as possible, and, hopefully, more usable.

Figure 5 shows the model structure proposed for this prototype. A general overview shows that it is composed of both PIM and PSM specific for Asyn drivers. The architecture proposed here is positioned in the meta-model layer of the MOF. Therefore, any kind of device can be modeled without changing this model. Each device, for example a vacuum pump controller is represented with an AsynDeviceSupport which extends the Device from the PIM. Any such device can contain any number of Commands (extended by AsynCommand) and properties, whilst commands themselves can contain any number of parameters. A property can contain one or no command for getting and one or no command for its setting.

CONCLUSION

It is the opinion of the author that MDA is the way development of control systems should take. Based on the prototype solution, choosing an approach offers more advantages than disadvantages. It offers an improvement in understanding the code – not only at development time but also later in the development process, in particular during maintenance. By using as many available tools, such as the EMF and MDDi from the Eclipse Foundation, and extending them to accommodate for different approaches required with building of control systems we were able to show, that a tool allowing transformations from abstract models of devices to concrete implementation of EPICS drivers is possible. Although the solution presented in the form of the MDA Support for Asyn Driver is not fully implemented, it will be developed and researched further.

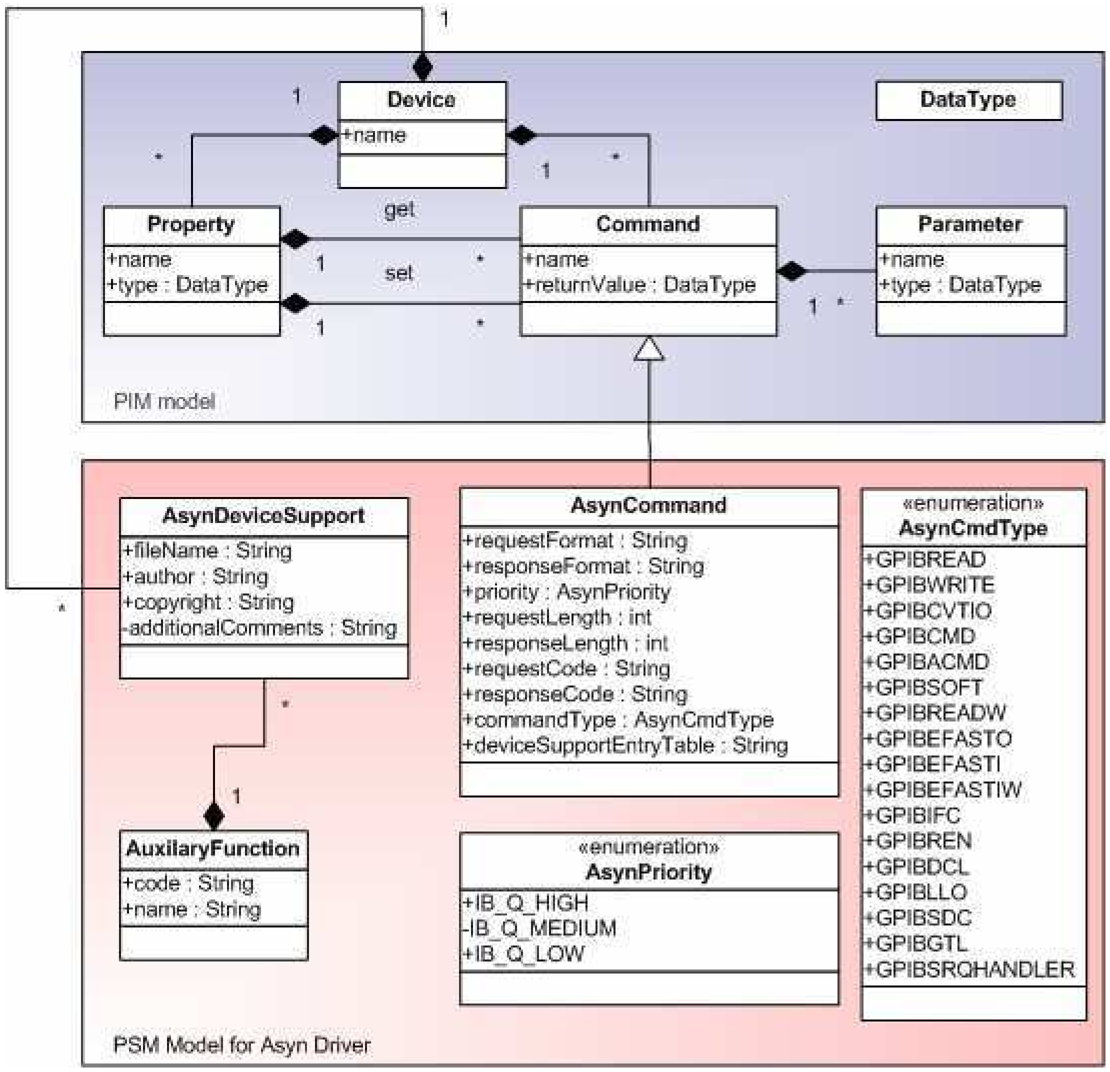


Figure 5 Meta-model of control system PIM (top) and a PSM (bottom), specific for EPICS asyn drivers.

REFERENCES

[1] Object Management Group – Model Driven Architecture, http://www.omg.org/mda/.

[2] Dan Matheson et al.**,** **"**Managed Evolution of a Model Driven Development Approach to Software-based Solutions**",** *OOPSLA & GPCE Workshop 2004.*

[3] K. Žagar et al., "The Control System Modeling Language", ICALEPCS'2001, San Jose, USA, October 2001.

[4] Eclipse IDE, www.eclipse.org

[5] Eclipse MDDi, http://www.eclipse.org/mddi/

[6] Eclipse GMT, http://www.eclipse.org/gmt/

[7] Eclipse UML2, http://www.eclipse.org/uml2/

[8] Eclipse EMF, http://www.eclipse.org/emf/

[9] Jon Siegel, "Making the case: OMG's Model Driven Architecture", SD Times, October 15, 2004

[10] OMG MOF Specifications, http://www.omg.org/docs/formal/02-04-03.pdf