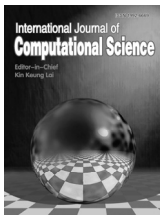
International Journal of Computational Science
1992-6669 (Print) 1992-6677 (Online) www.gip.hk/ijcs



Model-Driven Architecture Applied to Distributed Embedded System Design

Hiroyuki Mukasa[1], Takashi Shiraishi[2], Nurul Azma Zakaria[3], Noriko Matsumoto[3], Norihiko Yoshida[3]*, Toshihiro Nakayama[4]

[1] NEC Informatec Systems, Nakahara, Kawasaki 211-8666, Japan
h-mukasa@jp.nec.com
[2] Toshiba Solutions Corp, Musashidai, Fuchu 183-8532, Japan
Shiraishi. Takashi@toshiba-sol.co.jp
[3] Graduate School of Science and Engineering,
Saitama University, Saitama 338-8570, Japan
{azma,noriko,yoshida}@ss.ics.saitama-u.ac.jp
[4] Nippon Signal Co. Ltd., Kuki 346-0029, Japan
nkym-ts@signal.co.jp

Abstract. This paper presents some studies to prove applicability of the Model-Driven Architecture (MDA) technique to design of distributed embedded systems, or embedded system networks. MDA was originally introduced in the Software Engineering research, and has been widely accepted in software design as the technique encourages model-based component reuse, stepwise decision making, and early stage verification, all of which accelerate design processes. We first present an application of MDA to one of the largest distributed embedded systems: Automatic Train Control (ATC) systems. We then present its application to a design pattern for distributed embedded systems, to verify the applicability from a more abstract point of view. Successes in these works will benefit to accelerate design processes and improve product qualities of distributed embedded systems in general.

Keywords: distributed embedded systems, embedded system networks, Model-Driven Architecture, Executable UML, stepwise refinements, Automatic Train Control systems, design patterns.

* Corresponding Author. Email: yoshida@ss.ics.saitama-u.ac.jp.

1 Introduction

Distributed embedded systems, or embedded system networks, are getting popular in industry. They involve communication as well as hardware and software, and are more difficult to design than single embedded systems. To make design of distributed embedded systems easier, we are applying the Model-Driven Architecture (MDA) technique [1].

MDA was originally introduced in the Software Engineering research, and has been widely accepted in software design. It enables us model-level component reuse, which is beyond language-level reuse such as libraries, frameworks, and middleware. An abstract specification model is defined in the form of Executable UML (xUML) [2], and is transformed into a concrete implementation model with specific platform information and constraints, and then is translated into a program code.

Applicability of MDA to distributed embedded system design should be verified from several aspects. This paper presents two studies out of them: an application to a large system, and representation of design patterns for such systems.

An Automatic Train Control (ATC) system [3] is one of the largest distributed embedded systems. A track of a railroad is segmented into "block sections", each of which runs for several hundred meters. A controller is assigned to each block section, and cooperates with neighbors so that only one train runs in each section at a time, at a specified velocity at maximum. If a train comes closer to the preceding one, its velocity maximum is made lower, and eventually zero so that the system avoids train collision. This velocity maximum is notified to the train driver at a driving panel on the train.

All the ATC systems share a common specification and a design core, however their implementations vary according to actual constraints and requirements posed by customer railroad companies. Each implementation is currently developed in an ad-hoc fashion, which leads to an inefficient development process and poor product quality. Some techniques are strongly required for specification-level component reuse and implementation derivations.

We specify a simple ATC specification model in xUML, and transform it into an implementation model in a stepwise manner [4], following the System-Level Design (SLD) process [5,6]. We utilize our experiences on SLD applied to control systems [7,8] and to communication design [9,10].

The second part concerns with design patterns [11,12], which are descriptions of typical problems and solutions which frequently appear in various applications. The benefit of design patterns is to share solutions beyond applications, to reuse the solutions as well as to enable developers to communicate more easily. Design patterns have thus far been represented using the conventional (not executable) UML and sample codes in such programming language as Java, C++, or C#. From the language-independent point of view, they should be described using xUML. Distilled from some application implementations including the above, a design pattern for distributed systems is presented in xUML.

Through these two studies, we advocate our contribution to design process improvement of distributed embedded systems. The first one is being exploited practically in a company, and the second was verified in an experiment.

In the rest, Section 2 introduces MDA and xUML briefly. Section 3 then presents a simple ATC specification, and Section 4 describes its transformation. Section 5 presents what we name "Worker Sender" design pattern in xUML. Section 6 contains some concluding remarks.

2 MDA and xUML

MDA is to introduce transformational derivation from a "platform independent model" (PIM) specification to a corresponding "platform specific model" (PSM). A PIM defines the behaviors and functions of a system only, which are independent from its implementation platform such as any programming language, middleware and library. From the PIM and platform specific information, a PSM is derived which is to translated into a specific program code implementation. A different platform yields a different PSM from a single PIM, for example, Java-based, C++-based, or C#-based.

Each model is represented in the form of executable version of UML (Unified Modeling Language). The executable UML (xUML) adds well-defined execution semantics to the original UML so that model in xUML can be executed and verified. The core part of xUML comprises of the class diagram and the statechart diagram. The former defines the static aspect (the structure of the model), while the latter defines the dynamic aspect (the behavior of the model). The latter is described in the Action Specification Language.

The model transformation in MDA has close resemblance with stepwise refinements in the SLD process as shown in Fig. 1.

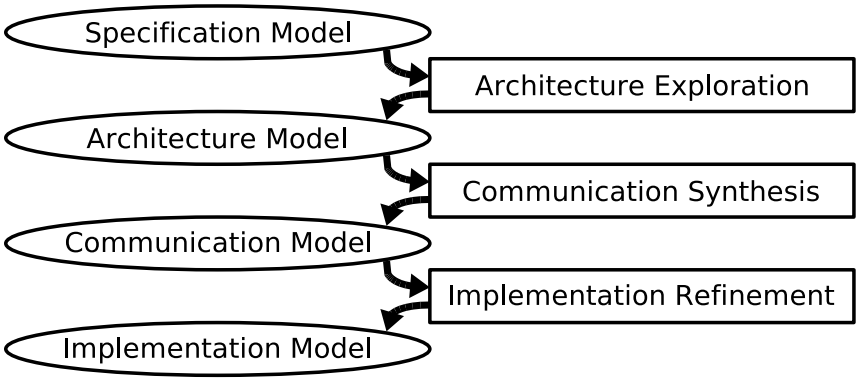


Fig. 1. Stepwise refinements in System-Level Design

3 ATC Specification Model

A track of a railroad is segmented into "block sections", each of which runs several hundred meters. A controller is assigned to each block section, and cooperates with neighbors so that only one

train runs in each section at a time, at a specified velocity at maximum. If a train comes closer to a preceding one, its velocity maximum is made lower, and eventually zero so that the system avoids train collision. This velocity maximum is notified to a train driver on a driving panel on the train.

This velocity maximum is defined in the chart of "aspect". The value is indexed with the position of the train, the position of the preceding train, the route, and a manual limiter. Therefore a schematic diagram of a single controller in the ATC system is composed as shown in Fig. 2. It receives the position of the preceding train from the controller at the preceding block section, and it sends the position of the current train to the controller at the following block section.

The class organization is specified in xUML as shown in Fig. 3. It is directly corresponding to Fig. 2. Each function or behavior is assigned to a class, without any consideration on concrete implementation.

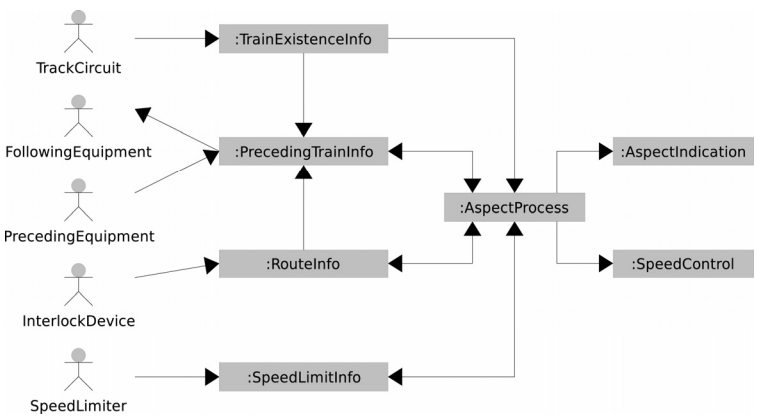


Fig. 2. Schematic diagram of ATC

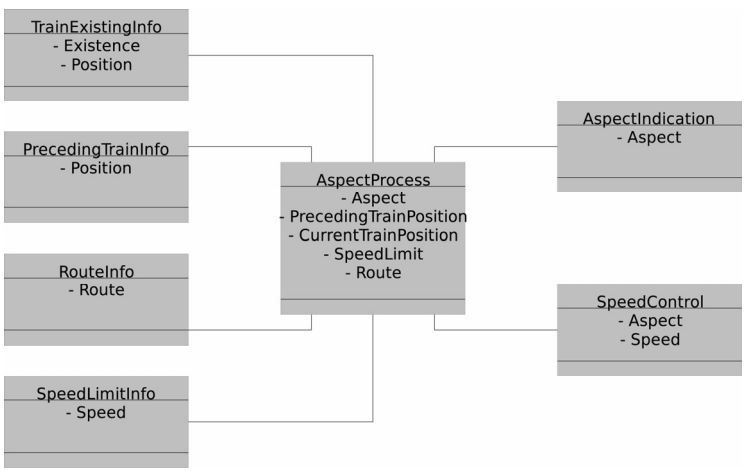


Fig. 3. xUML specification of ATC (1)

The state transition is specified in xUML as shown in Fig. 4. This is represented in Action Specification Language for xUML. The chart for position notification of preceding train is shown as an example. Each rounded rectangle represents a state, and codes in the rectangle represents action semantics at each state. Arrows represent state transitions. This sort of state transition definition is attached to every class presented in the class diagram.

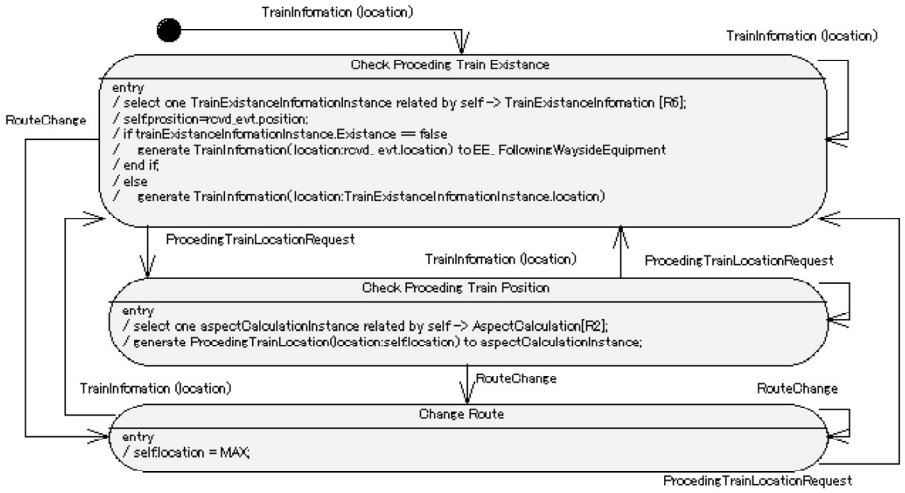


Fig. 4. xUML specification of ATC (2)

4 Transformational Derivation

The xUML specification is now transformed in a stepwise manner to derive implementation. We show this process using an example of communication part of the specification shown in Fig. 5. This is the same as the corresponding part appearing in Fig. 3.

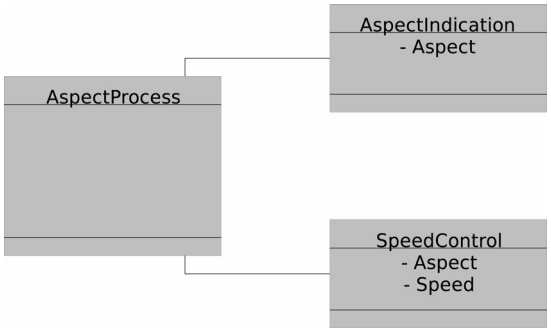


Fig. 5. xUML specification model

The specification includes three classes. The outputs of "AspectProcess" are transmitted to the control panel on the train for "AspectIndication" and "SpeedControl". This transmission is actually carried out with electric signal transmission over the railway.

First, this transmission scheme is transformed into a channel comprised of an abstract sender and a receiver. The sender-receiver correspondence is managed with the train position. Therefore, we add two new classes of "Sender" and "Receiver", both with the "position" attribute. This derived xUML definition is called an architecture model as shown in Fig. 6. What is important is that the functions and behaviors of the original specification model are strictly preserved, while its structure or architecture is modified so as to make the model a bit more concrete.

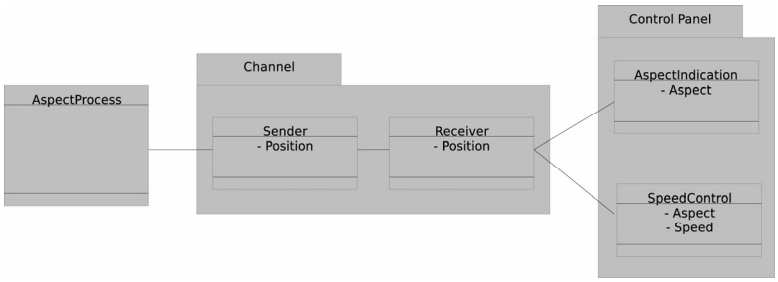


Fig. 6. xUML architecture model

Next, this abstract channel is transformed into a concrete transmission medium. The channel is replaced by two new components, "RailTransmittor" and "ReceiverUnit". The information message to send is carried in a modulated signal on the rail. The signal is then received by the receiver unit on the train, demodulated, and fed to the control panel. Therefore, some new classes, "Modulator", "Amplifier", and "Demodulator", are added to define this concrete implementation. Again, all the functions and behaviors of the model are strictly preserved. The derived definition is called a communication model as shown in Fig. 7.

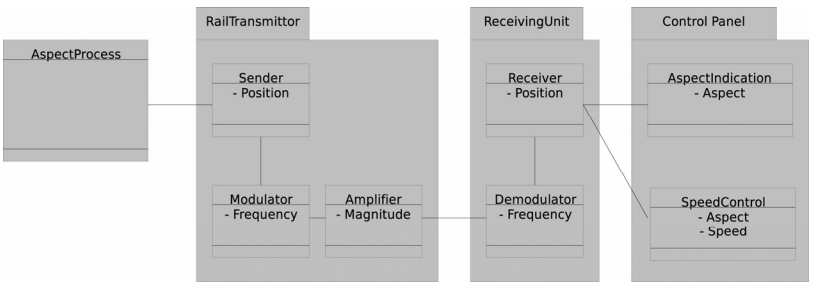


Fig. 7. xUML communication model

This communication model will be translated into a specific system-level language such as SpecC [13], SystemC [14], or SystemVerilog [15] for further implementation-specific transformation, and eventually its hardware part will be translated into some hardware description language

such as Verilog-HDL or VHDL, and its software part will be translated into some software programming language such as C, Java, C++, or C#.

Based on the xUML formalism, unlike traditional UML, all the models shown above are executable, and thus verifiable. We verified that all the models preserve the functions and behaviors of the original model. Our xUML platform is "iUML" [16,17], one of the xUML systems which are available to us. This stepwise transformation encourages model-based component reuse, stepwise decision making, and early stage verification, all of which accelerate design processes.

Nippon Signal Co. Ltd., to which one of us belongs, is one of the dominant companies in Japan engaged in designing, developing and manufacturing railroad control and operation systems. The above results have been out of our joint studies, and the company is now trying to exploit the results in actual design processes of ATC systems practically. This fact supports our claim that the results benefit to design and development activities in companies. However, we can only tell the fact, and cannot disclose details because of the agreement with the company.

5 "Worker Sender" Design Pattern

The "Worker Sender" design pattern is a distributed extension of the "Worker Thread" design pattern for single (not distributed) systems.

The "Worker Thread" design pattern is to handle request events in an on-demand and low-latency fashion. In short, a system based on this pattern pools some worker threads, and when it receives a request event from outside, it invokes an idle worker to handle the event.

The "Worker Sender" design pattern adds some functionality for forwarding, or "delegating" events among systems in a network in a hop-by-hop fashion. Such event operation must be important in distributed embedded systems, including the ATC systems as shown above, and even in general multi-server systems and web-service systems, for function distribution and load balancing.

The design pattern comprises of three abstract classes as main actors: Request, Channel, and Worker. Request is a capsule of an event, Channel is an event queue, scheduler and dispatcher, and Worker is a worker thread. There are also some auxiliary classes: Accepter, Sender, and Receiver specifies delegation and network communication.

Fig. 8 shows the class organization diagram of the "Worker Sender" pattern in xUML. Some abstract classes are divided into multiple classes: Channel is now composed of "Channel", "RequestQueue", and "ResultQueue" for example. Fig. 9 shows the state transition diagram of the class "RequestQueue" for example. Each of the other classes has its own state transition diagram as well. The class queues and assigns a "Request" event to a worker or delegate it to another system in the network.

We built a multi-server prototype model in xUML using this design pattern, and did an experiment. We observed that the system performs well in load distribution and balancing among multiple servers in a network. We also observed that we built this efficient system much easier than we would do without this design pattern, and we enjoy early stage verification in xUML.

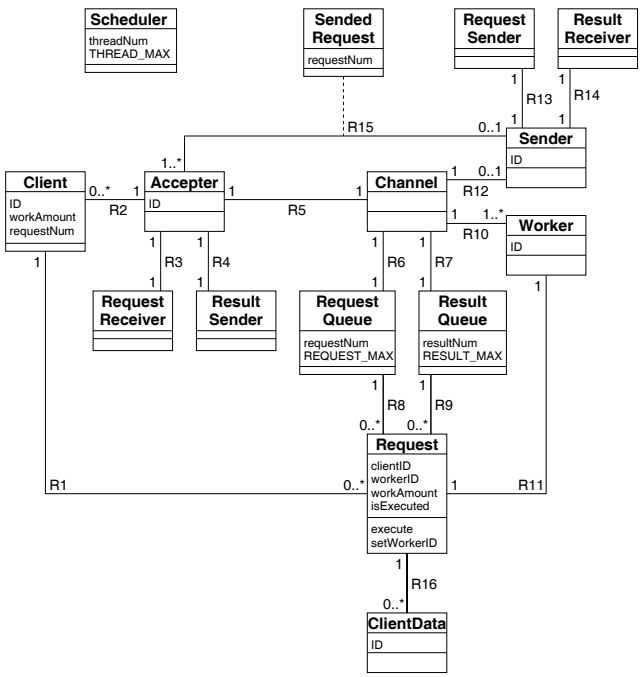


Fig. 8. Class diagram of "Worker Sender" pattern

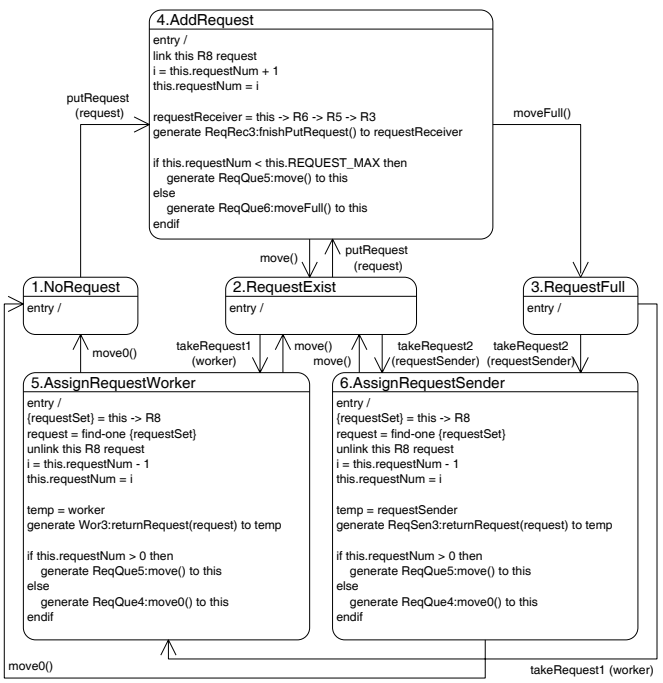


Fig. 9. State transition diagram of "RequestQueue" class

Design patterns in conventional UML-based representation are actually used for design catalogs, for designers' communication, and for novice training. Design patterns represented in xUML can be used as parts of PIM in MDA. Such design patterns act as implementation-independent specification models cooperating with other patterns and/or xUML-based components. This accelerates practical use of MDA. The xUML-based representation is also benefit to designers of design patterns, as it enables them to execute and verify correctness of new patterns. In this way, design patterns will be truly abstract components for system design.

6 Concluding Remarks

As far as we know, there has been no related studies to apply MDA and SLD to such large control network design as ATC. Also, this is almost the first attempt to represent design patterns for distributed systems in xUML. Consequently, we suppose these works will benefit to accelerate design processes and improve product qualities of not only ATC systems, but railroad control systems, and distributed embedded systems in general.

Both the two studies presented here assume homogeneous networks; all the components and network media in a system are homogeneous and consistent. It is left for further study to tailor distributed embedded systems on a heterogeneous network.

The MDA technique still has some shortcomings. The most serious one is that it is potentially too abstract to handle physical aspects of hardware devices such as fine timing constraints. To address this, MDA may need be used with such programming language as C, or even an assembly language.

We are still at the starting point in this research, and now building an automated tool for stepwise transformation of xUML models.

Acknowledgments

The authors are grateful to Dr. Kazutaka Kobayashi (InterDesign Technologies Inc., Japan), Dr. Ryosuke Yamasaki (Fujitsu LSI Technology Ltd., Japan), and Mr. Masahiro Kimura (Toshiba Solutions Corp., Japan) for their valuable contribution.

This research was supported in part by Saitama University and Nippon Signal Co. Ltd.

References

1. W. M. Camp: *The Need of Automatic Train Control*, Kessinger. (2007)
2. S. J. Mellor, K. Scott, A. Uhl, D. Weise: *MDA Distilled: Principles of Model-Driven Architecture*, Addison-Wesley. (2004)

3. S. Mellor, M. J. Balcer: *Executable UML: A Foundation for Model-Driven Architecture*, Addison-Wesley. (2002)
4. T. Shiraishi, N. A. Zakaria, N. Matsumoto, N. Yoshida, T. Nakayama: Executable UML Specification of Automatic Train Control Systems and Its Stepwise Transformation, *Proc. 2008 Int. Symp. on Appl. Comp. and Comp. Sci.*, 128-131.
5. K. Keutzer, S. Malik, A. R. Newton, J. Rabaey, A. Sangiovanni-Vincentelli: System Level Design: Orthogonalization of Concerns and Platform Based Design, *IEEE Trans.* CAD-19 (2000) (12) 1523-1543.
6. D. D. Gajski, J. Zhu, R. Doemer, A. Gerstlauer, S. Zhao: *SpecC: Specification Language and Methodology*, Kluwer. (2000)
7. R. Yamasaki, K. Kobayashi, N. A. Zakaria, S. Narazaki, N. Yoshida: Refactoring-Based Stepwise Refinement in Abstract System-Level Design, *Lec. Notes in Comp. Sci.*, No.4096, 712-721, Springer. (2006)
8. K. Kobayashi, R. Yamasaki, M. Kimura, N. A. Zakaria, N. Yoshida, S. Narazaki: Stepwise Refinement Design from Abstract Specifications Applied to Sequence Control Systems, *Int. Trans. on Comp. Sci. Eng.* 33 (2006) (1) 21-32.
9. K. Kobayashi, T. Shiraishi, N. A. Zakaria, R. Yamasaki, N. Yoshida, S. Narazaki: Exploration of Communication Models in the Design of Distributed Embedded Systems, *IEEJ Trans. on Elec. Electr. Eng.* 2 (2007) (3) 402-404.
10. T. Kinoshima, K. Kobayashi, N. A. Zakaria, M. Kimura, N. Matsumoto, N. Yoshida: Communication Model Exploration for Distributed Embedded Systems and System Level Interpretations, *Lec. Notes in Comp. Sci.*, No.4809, 355-364, Springer. (2007)
11. E. Gamma, R. Helm, R. Johnson, J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley. (1994)
12. D. Lea: *Concurrent Programming in Java: Design Principles and Patterns*, Addison-Wesley. (1999)
13. SpecC Web Site: http://www.cecs.uci.edu/~specc/
14. SystemC Web Site: http://www.systemc.org/
15. SystemVerilog Web Site: http://www.systemverilog.org/
16. iUML Web Site: http://www.kc.com/
17. C. Raistrick, P. Francis, J. Wright, C. Carter, I. Wilkie: *Model Driven Architecture with Executable UML*, Cambridge University Press. (2004)