Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

A metamodeling approach for safe control software

T. Collonvillé, L. Thiry, J-M. Perronne, B. Thirion

Laboratoire MIPS, Université de Haute-Alsace, Mulhouse (France)

e-mail: {thomas.collonville, laurent.thiry, jean-marc.perronne, bernard.thirion}@uha.fr

Abstract: Control software development is not an easy task. Lots of works deal with this problem and suggest solutions but in specific context only. The problem is that these solutions are compartmentalized at the technical and theoretical levels. This paper suggests to use the emerging domain of Model Driven Engineering (MDE) to simplify the interoperability between the various domains and disciplines used in control software design. The approach proposed is illustrated with the development of a control software for a legged robot.

Keywords: Model Driven Engineering, Concurrent Systems, Supervisory Control, Model Checking.

1. INTRODUCTION

Control software development is currently a complex task: Development requires the integration of many disciplines, models and how-to for modeling, analysis or design. The design complexity is increased by the domain specificities required by its implementation. Therefore it is necessary to insist on the fact that nowadays these design constraints (W.A. Halang et al., 2006) lead the software systems to become:

- Heterogeneous: The realization of the software system needs the use of different languages, platforms and concepts.
- Reactive: The system evolves according to external stimuli.
- Concurrent: The system is made up of individual elements that communicate.
- Fault tolerant: The system must be able to run even though unexpected stimuli leading to errors occur.
- Embedded: The platforms used are often lightened, and usable resources are limited.
- Distributed: The application is deployed on different platforms over a network.

These issues involve a stricter reasoning and more formal methods. Traditionally, the design of these software, said to be safety critical, is based on the use of development cycles such as the V-Model, Spiral Model, as well as the use of Agile Software Development Methods like: Unified Process (UP), Rational Unified Process (RUP) and eXtreme Programming (XP); see Ambler and Jeffries (2002) or Tomayko and Scragg (2004). In these development cycles, design and validation alternate but the development cycle always begins with the specification followed by the design of the control software. This consists in the modeling of the system and its constraints taking into account the specificities of the application domain.

The problem put forward by Ricardo Sanz is the limitation of the standard approaches unable to cope with the increasing complexity in the development of critical software. He explains that it is difficult to integrate the various domains which are necessary to design control software; Sanz and Arzen (2003). In Lee et al. (2004), the authors deal with this problem: "Recently, most works on systems modeling and simulation have been mainly focused on either Continuous Variable Systems (CVS) or Discrete Event Systems (DES)".

Taking these aspects into account, it is therefore necessary to build tools, to improve the already existing ones and to suggest methods for the development of such systems. It is particularly important to make these different domains communicate so as to set up a more rational and more reliable control software design process.

In this context, Model Driven Engineering (MDE) suggests a multiparadigm approach with various modeling languages described by meta-models and model transformations to link together these languages and the associated tools. In the field of control software engineering dedicated to concurrent and reactive systems, the aim of our work is to establish relations between the various domains required by the development process. More precisely, the paper proposes a framework for safe control software based on MDE concepts with meta-models for supervisory control theory and for model-checking, and models' transformations between these two kinds of models.

This paper is divided into four parts. The first part is the introduction. The second part introduces the MDE concepts. The third part presents the proposed approach. In the fourth and last part, an example illustrating the rationalized development of a control software is presented. In particular, these parts present supervisory control and model checking with MDE concepts.

2. MDE CONCEPTS

Model Driven Engineering (MDE) provides a framework to capture and then to integrate modeling concepts and tools. This integration is possible thanks to the use of meta-models, most often represented by UML diagrams, and to the use of model transformations, corresponding to graph transformations. In order to achieve this goal, MDE relates the system, the model, the modeling language and the meta-model as follows (figure 1); Mellor et al. (2003).

- The modeling language defines a set of models which includes specific characteristics of a domain.
- The meta-model is a model of a modeling language; more precisely, a metamodel defines the syntax and the semantics of this language. A model conforms to a meta-model is a member of the set of all models specified by the meta-model.

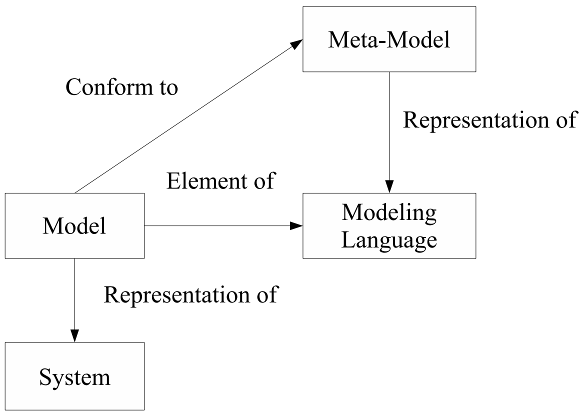

Fig. 1. Relations between the system, model, meta-model and modeling language

In order to link two domains, a meta-model has to be defined for each domain. It is then possible to associate the concepts of these different domains by the mean of transformation rules. These transformation rules enable to map one language to another language. This change in domains allows to take advantage of each of the modeling languages. The rules of transformation, applied to a model, build a new model which is conform to the meta-model of the targeted domain.

Mosterman and Vangheluwe (2000) propose a multi-paradigm approach based on MDE to model control systems. Lacoste-Julien et al. (2004) propose to use metamodeling concepts to define a hybrid formalism with AToM3 1 (see de Lara and H.Vangheluwe, 2002), a tool for metamodeling and model transformations.

MDE also provides tools (Eclipse EMF, Budinsky et al. (2003); GME, Ledeczi et al. (2001) or MetaEdit+, which can be configured with these meta-models and used in specific domains. In Rasse et al. (2005), A. Rasse uses for example MetaEdit+ to model dynamical systems and automatically get a representation that can be analyzed with a Model-Checking tool (in this case, LTSA, Magee and Kramer (1999)). Then, the model is transformed to obtain executable code.

3. PROPOSITION

To facilitate the development of control software, this paper proposes to apply the MDE concepts at each step of the development process with analysis, design checking, testing, etc. More precisely, the paper proposes to make software development easiest using MDE tools configured with meta-models and models transformations between domains which are necessary to system global design.

A similar approach has been proposed in Karsai et al. (2003) where the authors propose a model-integrated approach for embedded software development. Multiple views of a model are used in all steps of the software development cycle. This approach is called Model-Integrated Computing (MIC). However, this work is dedicated to embedded software and our approach proposes a development cycle model able to provide methods for a flexible approach of control software development.

The approach implies two kinds of actors. The first actor is the language architect who makes and integrates meta-models and transformations rules in MDE tools. The second actor is the system engineer who uses these pre-configured tool to model systems.

Thus, the language engineer does:

(1) The identification of the domains which are necessary to the analysis/design of control software.

(2) The definition of a meta-model for each of the identified domain.

(3) The refinement of these meta-models through a software component, which allows to provide a tool facilitating the design of models.

(4) The definition of transformation rules, by establishing mapping between the concepts of the various domains.

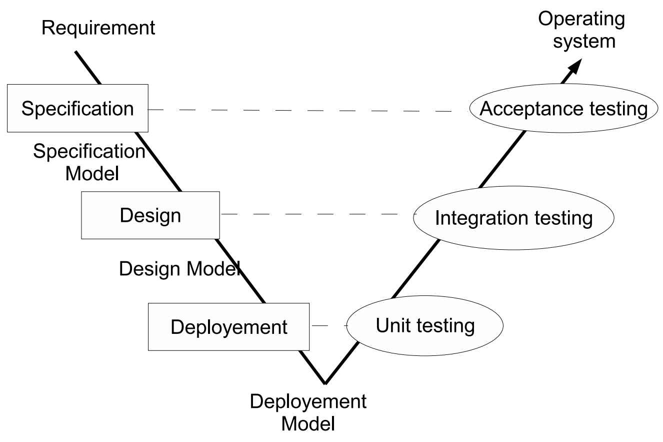

Fig. 2. V-Model development cycle

The aim of the previous work is to help the system engineer to apprehend the complexity of software development: in particularly, the system engineer must be able to use the tool pre-configured with various models (and meta-models) at each step of the design. For instance, the figure 2 describes the adaptation of a classical V-Model cycle to a V-Model cycle based on models. Each activity produces various models; and then there are models/meta-models (or modeling languages) for specification, design, deployment and verification. At this stage, model transformations have to be defined to allow passage from one activity to another.

4. EXAMPLE

4.1 Presentation

In order to illustrate the proposed approach, the paper studies the development of control software for a legged robot (figure 3.a). Further information about this platform, which is developed by our group, is given by Thirion and Thiry (2002). The control software is based on the parallel composition of six local controllers; each one describes the cycle of figure 3.b and the state machine of 3.c. The movement of a leg follows a cycle between two positions aep (anterior extreme position) and pep (posterior extreme position). A leg is said to be in retraction when it is on the ground and contributes to the movement of the platform. A leg is said to be in protraction when it is raised and moves towards aep. It waits until the event prot (for protraction) occurs. The simplified operating model of a leg is summed up in figure 3.c: aep and pep are uncontrollable events and prep is a controllable event.

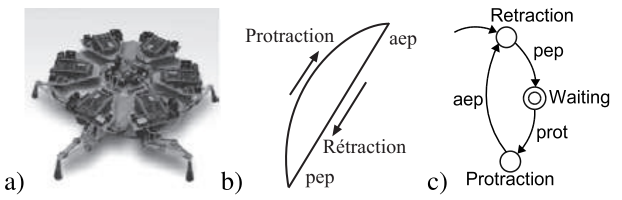

Fig. 3. a) Hexapod Robot, b) Leg operating cycle, c) Leg operating model.

It is necessary to set a few constraints/properties:

(1) The first constraint is required for the stability of the platform and states that two neighbouring legs cannot be raised simultaneously.

(2) The second constraint makes sure that the system always runs and states that the legs always follow their cycle: protraction, retraction, protraction, etc.

In order to facilitate the presentation, the paper will only focus on the study of two legs. The two constraints set up above imply that prot1, aep1, prot2 and aep2 have to follow one another. As pep1 and pep2 are only events indicating the end of the movement of the legs (1 and 2), they do not need any specific management.

The previous description gives the specification model of the system. Now, it is necessary to define how the control software has to be realized. The domain that appears in the development process has to be identified. As the specification uses discrete states, the main domains to be used are the one of Discrete Event Systems (DES) and the one of concurrent systems. More specifically, the two domains considered here are supervisory control theory, which allows to synthesize a first model of the control software and model checking, in order to validate the realization during the synthesis. In order to guarantee the equivalence of the model, the relation of bisimulation is to be used between the models of the supervisory control and those of the model checking.

4.2 Supervisory control

The first step of the realization consists in using the supervisory control theory to generate a model of the controller from the specification model. This proposition simplifies the realization but is not sufficient to make a system to satisfy all the constraints expressed by the specification; that is why Model Checking is used in the second step. Supervisory control is based on Discrete Event Systems (DES). It has been introduced by P.J. Ramadge and W.M. Wonham. It defines three entities: the model of the system G, the model of a wanted specification K and the model of the supervisor S.

The system is considered as an entity evolving in a autonomous way and generating events. It is coupled up with the supervisor and thus creates a control loop. The resulting behavior satisfies the constraints defined in the specification. At runtime, the system evolves as follows: at each step of the execution of the system G, the supervisor S is notified by the last generated event. Then, the supervisor notifies the system of the set of events it is authorized to. An event can be realized only if it corresponds to a physically feasible action and if it is authorized to do so by the supervisor, see Ramadge and Wonham (1989).

The models used for DES are based on language theory and on Finite State Machines (FSM); Cassandras and Lafortune (1999). The behavior of the system is therefore modeled by the language generated by an FSM with the following structure:

$G = (Q, \Sigma, \delta, q_0, Qm)$ where Q is a set of states, $Qm \subseteq Q$ is a set of marked states, Σ is a set of events called alphabet, q_0 is the initial state and d is the state transition function such as $\delta : Q \times \Sigma \rightarrow Q$.

Two languages can be distinguished: the language generated L(G) by the FSM, which corresponds to the possible traces (i.e. sequences of events) of a system, and the marked language Lm(G) which corresponds to all the traces ending into a marked state. (Note. $Lm(G) \subseteq L(G)$). The prefix closure of the marked language provides the set of traces Lm(G) and all the prefix of Lm(G) traces allowed. If $\overline{Lm(G)} = L(G)$ then it is guaranteed that the system G is non-blocking.

At this point, supervisory control provides the means of synthesizing the supervisor model from the model of the system and the model of the specification such as the coupling of the system G and the supervisor S satisfies the properties K. The coupling ensures that the language of the closed loop system is equal to the intersection of the language of the system and the specification so that $L(S/G) = L(G) \cap L(K)$. And so, the language L(S/G) corresponds to a model of behavior compatible with the activity of the system G and conforms to the specification. It is possible to design an algorithm for the automatic synthesis of the supervisor. The supervisor model is: $S=trim(G\times K)$, where trim is an operation deleting the blocking and non accessible states, and where $\times$ is the operation of strict product (Equivalent to $L(G) \cap L(K)$) between two machines G and K. The alphabet Σ is divided into two subsets: a set of controllable events Σ_C and a set of uncontrollable events Σ_U, so that $\Sigma = \Sigma_C \cup \Sigma_U$. Events in the set Σ_U cannot be forbidden by the supervisor. This restriction has to be taken into account for the synthesis of the supervisor: if the supervisor were built without taking controllability into account, it would be likely to forbid a non controllable event.

The Kumar algorithm takes this into account; it prohibits the controllable events leading to the uncontrollable events banned by the specification; see Brandt et al. (1990). The drawback of this solution is that it can delete states leading to behaviors necessary to the successful operating of the system. Indeed, the supervisory control guarantees the absence of unwished behaviors, but it does not guarantee the presence of particular behaviors. To face this problem, the domain of Model Checking provides tools and methods to check if the system is still satisfying the desired properties.

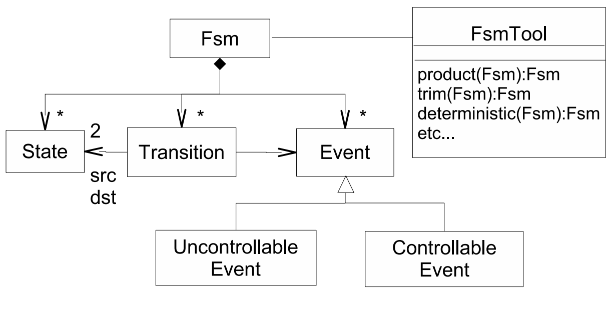

Fig. 4. DestKit: Meta-model for Finite State Machines (see definition G)

To validate our proposition, i.e. simplify the development of control software using MDE concepts, we have developed a Java framework (DestKit) for the modeling of DES and the synthesis of supervisor controllers. The framework relies on a meta-model, described in figure 4, which enables to describe and then transform Finite State Machines. DestKit integrates all the transformations necessary to the synthesis of a supervisor with a Finite State Machine describing a non-constrained system and another finite state machine describing the properties to be fulfilled. Moreover, it integrates classical operations on finite state machines such as different products (product or synchronized composition), the transformation of a non deterministic FSM into a deterministic FSM, the trim operation, etc.

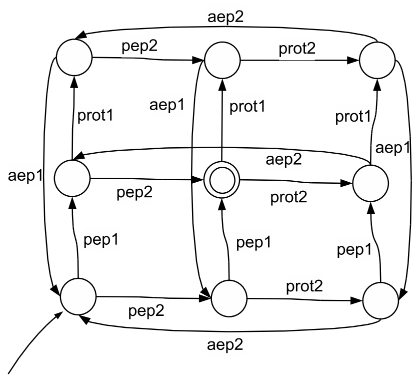

Fig. 5. Permissive model of a two leg moving cycle.

Thanks to DestKit, the two legs of the robot are modeled in their more permissive behavior by the product of the FSM, describing the behavior of each leg (figure 5). A specification is also defined in order to described the movement of the two legs (figure 6). It is then enriched by transformation: in adding uncontrollable events pep1 and pep2.

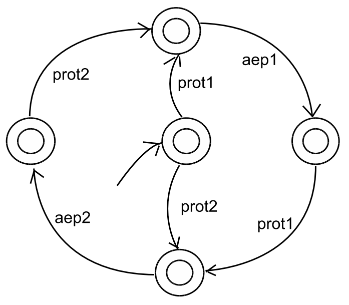

Fig. 6. Stability specification for the hexapod robot.

The supervisor (figure 7) is obtained by applying the synthesis algorithm and the Kumar algorithm. However, as the Kumar algorithm may give a more restrictive supervisor, it is important to check if the supervisor still fulfills some properties such as liveness. That is why it is necessary to switch to the domain of model checking; to check that the required properties are still satisfied.

4.3 Model-Checking

Model Checking is based on three steps: the modeling of the system M, the specification of the properties K and the use of an algorithm of model checking which checks whether the model M satisfies K: $M \models K$. The modeling step is generally based on Labelled Transition Systems (LTS), process algebra or FSM; see E.M. Clarke at al.

(2000). The specification of properties (such as liveness or safety) uses Linear Temporal Logic (LTL), Computation Tree Logic (CTL), LTS or FSM. The phase of verification is realized thanks to tools which put forwards the behaviors of the model which violate the properties. These tools usually provide traces leading to an unwished situation.

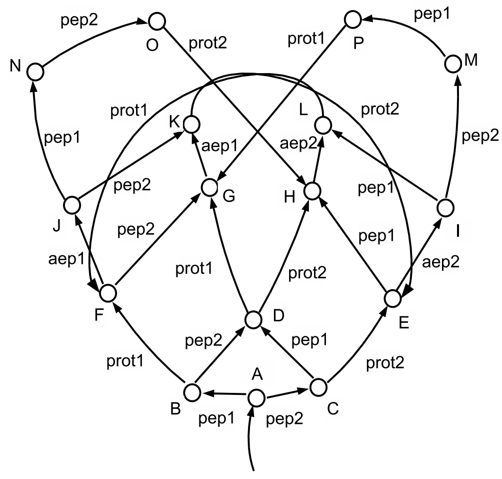

Fig. 7. Model of the supervisor

Verification used in our approach is similar to the one implemented in LTSA (J. Magee and J. Kramer, 1999): a model of the system M and a model of property K are expressed in the process algebra FSP (Finite State Processes); this later is a language to specify LTS models and LTL formulae. A simplified grammar for this language is presented by figure 8. A process can be terminated (STOP) or blocked (ERROR). A process executes an action a and becomes another process P noted (a→P). A process can choose between two actions (|). A process can be defined by the parallel composition of two processes (||). In order to check the fulfillment of the properties, LTSA produces the complement $\overline{K}$ of the property. So, $M \models K$ if and only if $L(M) \cap L(\overline{K}) = \{\}$. After verification, LTSA automatically generates a possible trace leading to an unwished configuration. It also allows the user to simulate/test the evolution of the system.

In the example of the hexapod robot, the validation of the models requires the transformation of the system and of the supervisor models into FSP to be used by LTSA. This transformation is illustrated on figure 9; it associates the key concepts of the metamodel of the supervisory control (figure 4), and the key concepts of the metamodel of model checking. Here the metamodel is represented textually by the grammar of figure 8.

```
Process   :: STOP
           | ERROR
           | Action → Process
           | (Process||Process)
           | Process | Process
```

Fig. 8. Simplified abstract grammar of FSP

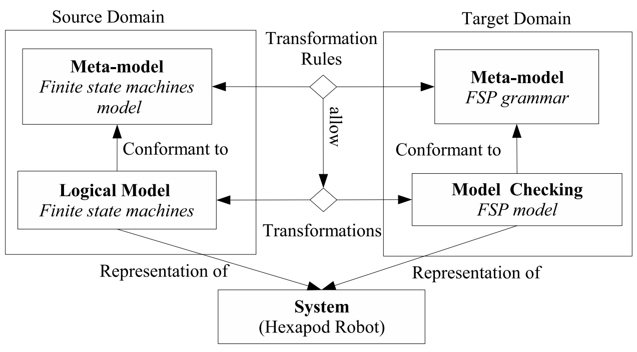

Fig. 9. Model of transformation FSM to FSP

So, an FSM turns into a process, an event turns into an action, etc. The result of this transformation provides the model of the supervised system, figure 10. On this figure, the property of liveness for two legs is described in FSP.

```
Hexapode =A
A=(pep1->B|pep2->C),            B=(pep2->D|prot1->E),
C=(pep1->D|prot2->F),           D=(prot1->G|prot2->H),
…
progress P1={prot1}
progress P2={prot2}
||CHECK=(HEXAPODE||Prop).
```

Fig. 10. Extract of the FSP model of the legged robot and liveness property

In order to validate the model transformations between the domain of supervisory control and the domain of model checking, the equivalence of the corresponding models has to be proved. This can be done using the relation of bisimulation which allows a stricter equivalence than the one based on the languages; see R. Milner, 1989.

Definition 1. Given two LTS $G=<S,T,\alpha,\beta,\gamma>$ and $G'=<S',T',\alpha',\beta',\gamma'>$ with the same alphabet Σ and where S and S' are sets of states, T and T' are transition functions, α and α' are functions giving source states of a transition, β and β' are functions giving target states of a transition and γ and γ' are functions giving event of the transition. The bisimulation relation is a binary relation $\mathcal{R} \subseteq S \times S'$ such that:

1. $\forall s \in S$, $\exists s' \in S'$ such that $s \mathcal{R} s'$
2. $\forall s' \in S'$, $\exists s \in S$ such that $s \mathcal{R} s'$
3. $\forall t \in T$ and $\forall s' \in S'$ such that $\alpha(t) \mathcal{R} s'$, $\exists t' \in T'$ such that $s'= \alpha(t')$, $\gamma(t)= \gamma'(t')$ and $\beta(t) \mathcal{R} \beta'(t')$
4. $\forall t' \in T'$ and $\forall s \in S$ such that $s \mathcal{R} \alpha'(t')$, $\exists t \in T$ such that $s= \alpha(t)$, $\gamma(t)= \gamma'(t')$ and $\beta(t) \mathcal{R} \beta'(t')$

As Finite State Machines are specific LTS, it is possible to establish the relation of bisimulation between the different states of the supervisor model (figure 7) and the different states of model obtained after the transformation in the domain of model-checking (figure 11).

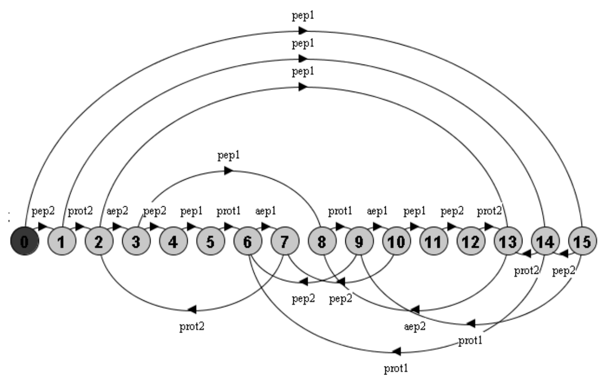

Fig. 11. LTS model of the supervisor resulting from the transformation (figure 9)

To conclude, the analysis of the FSP code introduced in figure 10 with LTSA, shows that specifications are satisfied. The model of the control system, coming from the generation of the supervisor at the previous step, satisfies the liveness property; the system can still generate the prot1 and prot2 events infinitely often. The design is done, checked; and what remains to do is to deploy the system on a platform.

5. CONCLUSION

Based on the example of an hexapod robot, this paper proposes an approach using the concepts of the Model Driven Engineering (MDE) community to make the development of control software easier. More particularly, this paper shows how to integrate specification models with design models (FSM and LTS). This integration is based on synthesis (with supervisory control theory) and checking (with model checking). The approach can be generalized to integrate other modeling languages, and tools, that appear in the analysis/design process. As shown in the example, the control software was created by using domains such as supervisory control (for synthesis) and model checking (for validation). This design therefore allows a unified conception of software systems, in a more reliable way. So, this approach allows a simplification of the process of the software development thanks to the integration of models allowing to: (1) Realize models at a higher level of abstraction. (2) Associate various domains by establishing rules of transformation between the various metamodels.

Besides, the perspectives of these works, it is important to define which domains can be used in a single way. In the same manner, it is also important to determine in which order they have to be used, which may imply a hierarchy among the domains considered in the development process.

REFERENCES

Halang W.A., Sanz R., Babuska R., Roth H., *Information and Communication Technology embraces Control*, Annual Reviews in Control, Vol. 30, pp. 31-40, 2006

Ambler S.W. and Jeffries R., *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. Wiley, 2002.

Tomayko J. and Scragg G., Human Aspects of Software Engineering. Charles River Media, 1er edition, 2004.

Sanz R. and Arzen K.-E., Trends in software and control. IEEE Control Systems Magazine, 23:12-15, 2003.

Lee J.-S., Zhou M .-C, and Hsu P.-L., A multi-paradigm modeling approach for hybrid dynamic systems. *IEEE International Symposium on Computer Aided Control Systems Design*, pages 77-82, 2004.

Mellor S.J., Clack A.N., and Fatagami T., Model-driven development - guest editor's introduction. *IEEE Software*, 20:14-18, 2003.

Mosterman P.J. and Vangheluwe H., Computer automated multi-paradigm modeling in control system design. In *Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on*, pages 65-70, 25-27 Sept. 2000.

Lacoste-Julien S., Vangheluwe H., De Lara J., and Mosterman P.J., Meta-modelling hybrid formalisms. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 65-70, 2004.

De Lara J. and Vangheluwe H., Atom3: A tool for multi-formalism modelling and meta-modelling. In Springer- Verlag, editor, *European Joint Conference on Theory And Practice of Software (ETAPS), Fundamental Ap- proaches to Software Engineering (FASE)*, pages 174-188, Grenoble, France, April 2002.

Budinsky F., Steinberg D., Merks E., Ellersick R., and Grose T.J., *Eclipse Modeling Framework*. The Eclipse Series. Addison Wesley Professional, 2003.

Ledeczi A., Maroti M., Bakay A., Karsai G., Garrett J., Thomason C., Nordstrom G., Sprinkle J., and Volgyesi P., The generic modeling environment. In *IEEE In-ternational Workshop on Intelligent Signal Processing*, 2001.

Rasse A., Perronne J.M., and Thirion B., Using process algebra to validate behavioral aspects of object ori-ented models. In *Models/UML'2005, Modeva Workshop*, page 10, Montego bay, Jamaique, 2005.

Magee J. and Kramer J., *Concurrency: state models & java programs*. John Wiley & Sons, Chichester, 1999.

Thirion B. and Thiry L., Concurrent programing for the control of hexapod walking. ACM SIGAda Ada Letters, 22:17-28, 2002.

Ramadge P.J. and Wonham W.M., The control of discrete event systems. IEEE Transactions on Automatic Control, 77(1):81-98, 1989.

Cassandras C. G. and Lafortune S., Introduction to discrete event systems. Springer, New York, 1999.

Brandt R. D., Garg V. K., Kumar R., Lin F., Marcus S. I., and Wonham W.M., Formulas for calculating supremal controllable and normal sublanguages. Systems and Control Letters, 15(8):111-117, 1990.

Clarke E.M., Grumberg O., Peled D.A., *Model Checking*, The MIT Press, Cambridge, 2000

Milner R., *Communication and Concurrency*, Prentice Hall International Series in Computer Science, 1989