A MODEL MANAGEMENT APPROACH FOR CO-SIMULATION MODEL EVOLUTION

Xiaochen Zhang and Jan F. Broenink
Centre for Telematics and Information Technology, University of Twente, Enschede, The Netherlands

Keywords: Multi-disciplinary system, Model management, Co-simulation, Model evolution, Logging mechanisms.

Abstract: In most of the embedded control system designs, multiple engineering disciplines and various domain-specific models are involved, such as mechanical, control, software and hardware models. Close collaboration and well integration between all domain-specific models become more and more important for developing dependable and cost-efficient systems. Moreover, each disciplinary model can be developed and evolved following its own semantics and development tools in different rates. The inconsistency between models and the evolution of the collection of models will increase the complexity of design as well as the difficulty of maintaining several models under simultaneous development and changes.
This paper proposes a model management approach for multi-disciplinary systems and co-simulation. Such model management approach can ensure the model integration and consistency by checking the model interfaces attached to each domain model and the protocol defined in a co-simulation contract. It also can keep track of model evolution along with changing details and making design variants. The concepts of a scenario-based co-simulation framework and a logging mechanism with graphical representation of the model evolution process are also explained.

1 INTRODUCTION

Simulating models is a widely used means for verifying the correctness of a system design. For most model-based embedded control systems, multiple engineering disciplines and various domain-specific models are involved. See, for example, the self-balancing human transport vehicle (Segway, 2007) shown in Figure 1, in which the system design requires mechanical models, hardware and software models, plant model and its control algorithms, as well as the requirements and a project/product management system to keep the design on track.

However, even if working on the same system and towards the same overall goal, engineers from different domains intend to use their domain-specific languages and tools to interpret the problems, and focus on that specific part. For instance, an expert from the control engineering domain decides to use 20-sim (Controllab Products, 2010) to simulate the dynamic behaviours of the system. On the other hand, a software engineer intends to use the Overture Tool (Community, 2010) to implement the logic behaviour of the system. The traditional way of implementing the system is to follow a mono-disciplinary style (Fitzgerald et al., 2010), in which, each engineering group handles distinct aspects separately and provide integration after all aspects are finished. Unfortunately, this late integration and less support of system-level overview may cause fatal problems since the impact

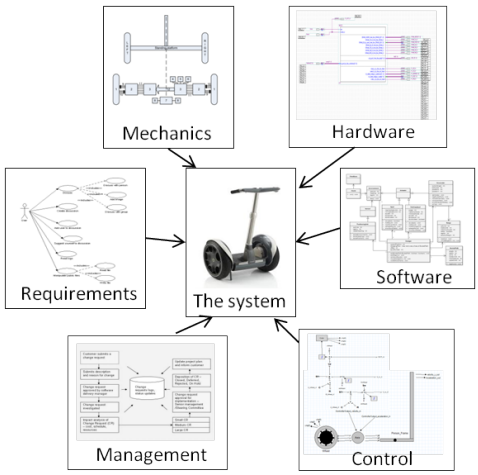

Figure 1: An example of possible engineering disciplines required for a self-balancing scooter system (Kuppeveld, 2007).

Zhang X. and F. Broenink J..
A MODEL MANAGEMENT APPROACH FOR CO-SIMULATION MODEL EVOLUTION.
DOI: 10.5220/0003601201680173
In *Proceedings of 1st International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2011), pages 168-173
ISBN: 978-989-8425-78-2

on each discipline is usually exposed late in integration phase. To support multi-disciplinary system design, we advocate collaborative modelling and co-simulation (DESTECS, 2010), in which, systems as a whole can be verified in the early stage of their lifecycle.

In our co-simulation perspective, all models can be developed and evolved concurrently following their own semantics and tools in different rates. The evolution of models naturally occurs during the design process, but models also evolve due to changes in requirements. In particular, changes in one model may have ramifications on other models since all domain models are somehow dependent on each other. The mismatch of interconnection of co-simulation models will cause incorrect simulation results. Moreover, the lack of an overall view of a co-simulation model development process will also slow down the design speed and the possibility of reusing models. All these aspects will highly increase the complexity of system design as well as the difficulty of maintaining co-simulation models while evolving.

Hence, our work is aiming at addressing a co-simulation-oriented model management approach to ensure model integration and consistency, and prevent mismatches of model parts. The second goal of the approach is to keep track of model evolution, such that various levels of detail and maturity can then be used to find optimal solutions for design challenges, and support decision making. These services need to be implemented in tools to manage complexity, structure the modelling process, and manage concurrent design.

In this paper, Section 2 provides the key concepts and model consistency checking method by introducing the terms *model interface* and *co-simulation contract*. Section 3 presents the model management architecture overview and its main services. Section 4 explains the two-directional model evolution approach, and addresses a graphical representation of the model evolution process and the supporting tool. The final section gives the concluding remarks and a forward look to the future.

2 KEY CONCEPTS AND MODEL CONSISTENCY CHECKING

Models are an important organisational resource (Chao-fan and Qing-Yan, 2009). Since the constitute models of a co-simulation can evolve in different speeds through development, the success of running a co-simulation is supported by synchronising domain models in a consistent fashion. In this section, the idea of scenario-based co-simulation is explained. The definition of co-model and model interface are introduced as well to ensure the model integration and consistency checking.

2.1 Scenario-based Co-simulation

In order to execute a co-simulation run, a test scenario is required to drive the process. A simple scenario can be thought of as defining a test run for the co-simulated models. A complex scenario may configure multiple co-simulation runs, for example, a parameter in a model is defined within a range to allow parameter sweeping. Engineers can define different test scenarios for the same co-simulation models to verify and test the system behaviours, and furthermore, the dependability of the system under extreme situations.

The scenario is linked to the domain models via a so-called *co-model interface*, shown in Figure 2. The co-simulation then can be executed by executing the domain models according to the co-simulation configuration defined in the scenario, such as the initial values of certain shared variables and design parameters, or defining system behaviours that should occur at specific points in time during the co-simulation run.

2.2 Co-model and Model Interface

Models are normally executable and are the foundation elements in model-based engineering. The concept of a *co-model* is introduced to integrate different domain models and exchange data for co-simulation. The interaction between domain models is achieved by executing them simultaneously and allowing information to be shared. The sharable elements are recorded in a co-simulation *contract*. Each domain model is connected to the contract by attaching a *model interface*. The structure of a co-model is shown in Figure 2, in which the co-model is comprised of a discrete-event (DE) model, a continuous-time (CT) model, and the co-simulation contract. In this way, the co-simulation can be verified by checking whether the shared elements returned from each model interface match the statements written in the contract.

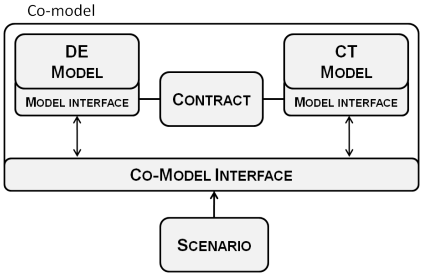

Figure 2: Co-model structure and scenario-based co-simulation.

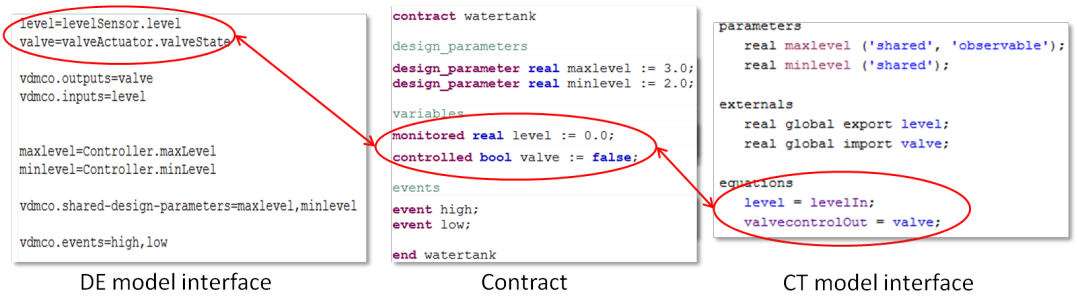

Figure 3: Example of model consistency checking in a water level control system design.

2.3 Consistency Checking

The consistency and compatibility of the shared information are important for co-simulation. Once the interface of the models are changed, the contract has to be modified as well. The model interface for each of the constituent model can be accessed externally.

Figure 3 shows an example of how the constituent model interfaces connect to the contract and ensure the consistency in a water level control system design. The left-hand side is a discrete-event model interface, in which the `levelSensor.level` and the `valveActuator.valveState` are connected to the contract by giving their values to those shared variables defined in the co-simulation contract, i.e. the middle code block. The right-hand side is the continuous-time model interface, in which the `levelIn` and `valvecontrolOut` are connected to the contract as well through the same identifier.

These interfaces are intended to assist in managing the models by permitting a degree of static checking on the viability of co-simulation. This ensures model coherence and consistency, also while the models evolve. However, the level of information hiding is currently limited. For example, interface-preserving changes might nevertheless invalidate the semantics of a co-model and this only becomes visible at co-simulation time. This is, unfortunately, not yet covered by the consistency checking system.

3 MODEL MANAGEMENT ARCHITECTURE

A traditional model management system usually contains several services such as storage and retrieval of models, handling of versions and variants of models, access control, change request management, development process management, analysis of the results of the model if necessary, as well as support for geographically distributed development (Baharadwaj et al., 1992). Since we are working with many models within a cross-domain project, more services of the model management system should be provided. This section presents the model management architecture and its basic components.

In order to fulfil the demands of a co-simulation development, the model management architecture has to include at least the following three aspects:

The first and primary aspect of the model management system in our approach is to keep track of the model development and evolution process of the system design in order to deal with the increasing complexity introduced by continuous change of models. The second aspect is the service towards model interfacing, integration, and consistency checking. The third aspect is the model base, and its ability to store, share and exchange data information for co-simulation purpose.

Together with the components mentioned above, the proposed overview of the model management architecture is illustrated in Figure 4.

The architecture consists of two main parts: a set of functional facilities and a model base. The model base can be considered as a model storage system. Models in the model base can be accessed by each of the domain-specific tools. The detailed functionalities of each components in the architecture are listed here:

- Model Base: to share and exchange model information that are required by a co-simulation. The model base can be considered as a data repository. Models that are stored in the model base are the first step towards model reuse.
- Model Integration: to integrate different domain models such that designs can be treated and managed as a whole; the concept of co-model is introduced by this function.
- Consistency Checking: to check the consistency and compatibility of constitute models within a co-model (via the model interface and contract introduced in Section 2), and model alternatives within a model base.
- Model Evolution: to support the model evolution

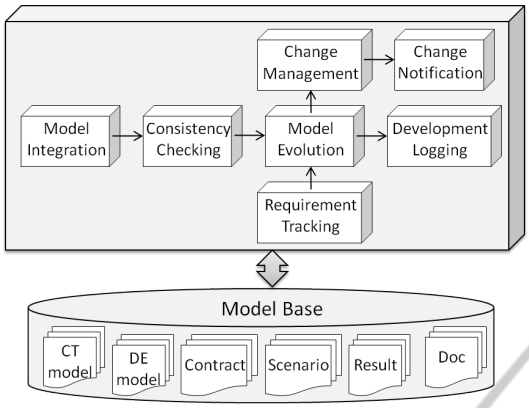

Figure 4: Overview architecture of the model management approach.

along 2 directions (versions and variants); provides a top-level overview of the entire developing process.

- Requirement Tracking: to keep track of requirements and its changes throughout the development.
- Change Management: to keep track of the process of model evolution, including model version control. An automatic way of detecting changes on all domain models and the method of comparing model topology and topography differences will be designed for this service as a part of the model management system.
- Change Notification: to inform the corresponding engineer who is assigned to an authorised model with notification messages when the model is changed by other engineers.
- Development Logging: to provide a graphical representation of the model evolution process.

4 MODEL EVOLUTION

In most complex system developments, requirements are vague in the beginning and are refined stepwise towards a more formal representation (Kim and Carrington, 2006). The frequently changing requirements lead to continuously evolving models. In practice, the maturity of a model grows by adding detailed information during the design. On the other hand, it is also common that engineers intend to try another design solution which is similar to an existing one and see which design can maximally fulfil the design requirements and requires less cost.

In this section, we present an approach of model evolution along two axes and how this approach, implemented as a tool, helps to handle the complexity of the co-simulation design process.

Technically, all these models are closely related and have to be stored in the model base such that the complete model evolution process can be tracked. Some of these model evolutions will survive design choice while others are abandoned, according to conclusions drawn from co-simulation results.

4.1 Model Evolution in Two Directions

In practice, it is common that more than one candidate design is made for the same development purpose in order to find the best design solution. Results are many versions of models for the same item of which some are refinements and others are different design variants, that is, design alternatives. For managing co-models, in our terminology, as they developed through a system's design history, we distinguish between alternatives and details. This can be seen as model evolution along two axes. *Design alternatives* are co-models which represent different solutions to the same problem. They generally differ from one another in terms of values of parameters, variables and structure. Taking the water level control system as an example, suppose that a motor is controlled by the local controller, and used to open and close a valve. Different types of motors might yield different results according to the capabilities of that motor with respect to speed, power or torque, each being an alternative model. Model variants can be created by mapping from other model alternatives.

Design details represent refinements of the same basic co-model. Each alternative can be refined through many different detailed levels so that each level represents a design step of a given alternative. Different than just keep track of all changes on a model, the detail level is considered as an important change status of the development. In stead of showing all changes of a model, including small and minor changes, providing a higher level view of the model evolution may help engineers to master the development process and design decisions based on models, with possibilities for future model reuse.

Through model evolution, several types of relations between models are classified to indicate the process:

- **Evolve** steps move a model towards its final form. This may entail the addition of detail to make a model more competent or more faithful to a proposed implementation. However, evolution steps can equally well involve the removal of irrelevant details from models, when it is determined that these do not contribute to the analysis for which

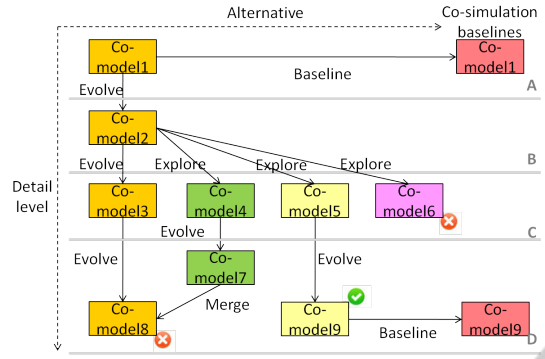

Figure 5: Model evolution process.

the model is constructed.

- **Explore** steps generate design alternatives; generation can be by hand or with machine assistance, based on the outcomes of design space exploration.
- **Merge** steps integrate and reconcile multiple changes into one design alternative.
- **Baseline** steps designate a particular model as significant in the development story. We might expect to see a full set of scenarios, scenario outcomes, rationale and documentation associated with such baselines.

Figure 5 illustrates, at an abstract level, an example of co-model evolution along detail levels and alternatives dimensions. It shows an extract of an imaginary model base in which each block stands for a co-model. The arrows indicate different types of the relationships between co-models in the structure, each with different attributes.

The horizontal grey lines show decision points, marked A to D in Figure 5. At each point, any decision to perform one of the actions above will be made, based on the design requirements and simulation results. For instance, before point A is reached, the original Co-model 1 is proposed. For documentation or configuration considerations, Co-model 1 is designated as a baseline. Through evolution, more components might be detailed out, such that the detail level of Co-model 2 is greater than that of Co-model 1. Between points B and C, four alternatives are explored based on the original model. Due to design space exploration, Co-model 6 may be considered not to fulfil certain design requirements as a result of applying relevant cost functions to its co-simulation test outcomes. As a result it can be designated a “dead end”. Other models may be a satisfactory basis for further exploration. In the example, Co-model 9 is ultimately designated to be the next baseline.

The above approach is to manage the co-model evolution life-cycle such that engineers can easily maintain the entire development process by navigating the model evolution history. All these co-models and their evolution process, the relations and the purpose of making the design decisions based on the models can be tracked.

Based on this approach, a readable document has to be generated to the engineers to tell the story of the co-model evolution process. Different detail levels and different co-model alternatives should be recorded and retrieved easily from the model base. Changes and decisions have to be recorded as well. Therefore, a tool prototype is developed as a proof-of-concept of this evolution approach.

4.2 Model Development Log and Tool Support

Development log files are valuable resources for designing, implementing, testing and documentation since developers can recall the process of the development according to the logged information. The idea of our logging mechanisms is to provide a graphical representation of the model evolution process in order to help understanding the development of especially large and complex embedded control systems.

Figure 6 shows an example of the basic logging mechanism. Engineers from different disciplinary domains can develop their responsible models concurrently. All modelling information and the simulation results can be stored in the model base. The model management system can take care of the model evolution process and the consistency checking. According to the development information stored in the model base, an top-level overview of the model evolution can then be generated in the “Development Log Graph” window which is implemented as an Eclipse plug-in, and shown in the right side of the figure.

All the blocks in the graph stand for co-models of different levels of detail in the model base, in this example, the water level control system model base. The block in green indicates that the co-model is a root model of the design alternative, also with a “root” mark. By default, the root co-model is marked as “important status” and “co-simulation baseline” indicating the starting point of this design.

In this example, there are two design alternatives in the entire development phase. The mark “E” on the co-model block indicates a design alternative, which is explored based on this design. The right-hand side shows the development information of the co-model, which appears when clicking on the co-model block. For example, the revision number, author information, date of design, decision messages of mak-

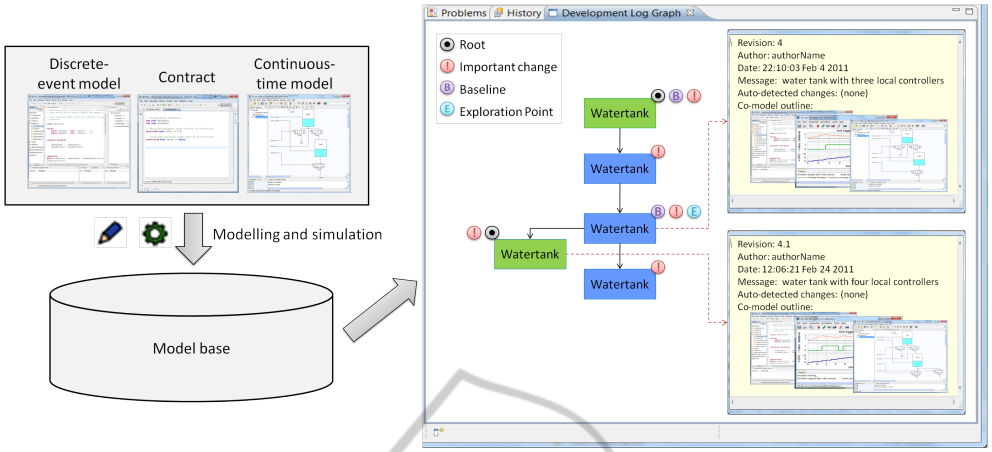

Figure 6: Graphical representation of model evolution logging mechanisms.

ing this design as an important change, the status of the design through evolution, and an outline of the co-simulation ingredients including the discrete-event model, continuous-time model and the results of running the co-simulation, are all illustrated. Additionally, engineers are allowed to navigating all detail changes of the evolution by zooming in the graph.

5 CONCLUSIONS

In this paper, a model management approach for multi-disciplinary modelling and co-simulation is proposed. This management approach is used to ensure the consistency between different domain models which can evolve in different rates during a system development. Based on the approach, a prototype system (DESTECS, 2011) in the form of mock-up has been developed to verify the proposed methods. A graphical representation of the co-simulation model evolution process can be generated and shown to the users, such that the overview of the model development process can be inspected.

The idea of auto-detecting changes on text-based and graph-based models is indicated in this paper, and its design is future work. Moreover, the complete implementation of this facility, including extensive testing is also future work.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 248134.

REFERENCES

Baharadwaj, A., Choobineh, J., Lo, A., and Shetty, B. (1992). Model management systems: A survey. In *Annals of Operations Research*, volume 38. Springer.

Chao-fan, D. and Qing-Yan, Z. (2009). Research and implementation of general model management and service system. In *International Conference on Management and Service Science, MASS '09*. IEEE.

Community, O. (2010). The overture tool website. http://www.overturetool.org.

Controllab Products (2010). The 20-sim dynamic modelling tool website. http://www.20sim.com.

DESTECS (2010). DESTECS Project Website. Website. http://www.destecs.org.

DESTECS (2011). The DESTECS SourceForge Project Website. Website. http://sourceforge.net/projects/destecs.

Kim, S.-K. and Carrington, D. (2006). A pattern-based model evolution approach. In *APSEC'06 XIII ASIA PACIFIC SOFTWARE ENGINEERING CONFERENCE*.

Kuppeveld, T. (2007). Model-based redesign of a self-balancing scooter. Master's thesis, University of Twente.

Segway (2007). Segway Personal Transporter. Website. http://www.segway.nl.