A Concurrent Design Approach and Model Management Support to Prevent Inconsistencies in Multidisciplinary Modelling and Simulation

Xiaochen Zhang, Jan F. Broenink
Robotics and Mechatronics
Centre for Telematics and Information Technology
University of Twente, Enschede, The Netherlands
E-mail: {x.c.zhang, j.f.broenink}@utwente.nl

KEYWORDS
Model management, model evolution, consistency checking, concurrent design flow, co-simulation

ABSTRACT
Cyber-physical systems are multidisciplinary systems which involve different engineering disciplines in their design. Each engineering discipline tends to use its own domain-specific languages and tools to model different aspects of a system concurrently. The concurrent modelling process may introduce inconsistencies due to lack of common knowledge and communication among domain experts. Especially for co-modelling and co-simulation developments, a huge amount of models, versions of models and design alternatives may be produced, which highly increases the design space and the chance of having inconsistent models. This paper introduces a model management support and concurrent design flow to prevent inconsistencies and maintain synchronization among models. Besides the consistency checking scheme, a co-evolution graph can be generated by the model management system to visualize the concurrent development process and prevent inconsistencies. The model management system and concurrent design flow have been applied on a line following robot to show how to use this approach and its advantages.

INTRODUCTION
Concurrent Engineering (CE) is a widely used methodology in system designs and involves engineers with various opportunities for communication. Especially for model-based cyber-physical system designs, the multidisciplinary nature implies that different engineering disciplines and domain-specific models are involved. In addition, close collaboration and coordination between different domain experts are essential to the success of such developments.

In traditional CE design flows, such as the Iterative Development Method (Larman and Basili 2003) and the Spiral Model (Boehm 1986), a physical prototype of the system is required to test and evaluate the design, which increases the time-to-market and design cost. Therefore, we advocate early integration and verification of different domain models via the co-simulation-in-the-loop workflow, shown as grey arrows in Figure 1. Variants of models can be implemented and tested after designs are verified by the co-simulation, in order to minimize the time and cost on producing intermediate prototypes.

A co-simulation framework, called DESTECS\(^1\), is used in this paper to provide a platform to co-simulate domain models in an early stage of development. With the help of DESTECS, different disciplinary models (i.e. discrete-event (DE) model and continuous-time (CT) model in a cyber-physical system), created using different domain-specific languages and tools, can be simulated and verified before real prototypes are made.

Taking a robotic arm design as an example, experts from the control engineering and mechanical domain use the 20-sim tool\(^2\) to model the dynamic behaviour (CT domain) of the robotic arm, while software engineers use the Overture tool\(^3\) to implement the real-time software

---
\(^1\)www.destecs.org
\(^2\)www.20sim.com
\(^3\)www.overturetool.org
logic (DE domain). A synchronisation scheme behind the co-simulation (Fitzgerald et al. 2012) takes care of the simultaneous execution of two tools and keeps their local simulation time synchronised.

Many versions of models and model alternatives produced at different levels of abstraction are related and have dependencies on each other. Although the DE model and CT model represent distinct aspects of the same system, overlaps on shared information still exist. Therefore, a need arises to check and ensure model consistencies during the entire co-simulation development cycle.

Unlike managing models representing different viewpoints of the same system, such as methods described in (Stumptner and Schrefl 2000), (Wang et al. 2010) and (Giese et al. 2010), finding a common formal specification like OCL (Object Constraint Language) between the DE model and CT model is difficult since they are developed and evolved simultaneously following their own semantics. Hence, inconsistencies between two domain models are mostly introduced due of the lack of common modelling knowledge and the concurrent evolution process.

Traditional approaches to the problem of concurrent working use a shared repository to which all team members have common access. Inconsistencies are usually managed through strict access control and version management, along with a common data model or schema (Nuseibeh et al. 1994).

In this paper, we propose a model management and consistency management approach, from the concurrent evolution of models point of view, to overcome the inconsistency issues in order to support the design of dependable mechatronics systems using the co-simulation technique.

The structure of this paper is as follows: we first give a brief overview of the co-simulation framework and key concepts. Based on this framework, the Model Management System (MMS) and its tool support are introduced, which describes the meta-data binding and the model evolution techniques, in order to help keep models organized and prevent inconsistencies. A case study section then follows which explains general guidelines and design flow of using the model management support. The final section gives the concluding remarks of the paper and a look forward.

**BACKGROUND**

The concept of a co-model is introduced to integrate DE and CT models and exchange data for co-simulation. As illustrated in Figure 2, a co-model consists of one DE model, one CT model and a co-simulation contract. All shared information between DE and CT models, such as shared variables (svars), shared design parameters (sdps) and events (evts), must be defined in the contract. Each domain model is connected to the contract by attaching to a model interface which provides a set of link statements to connect shared properties defined in the contract and in domain models, such that only shared properties are accessible to the counterpart model.

For each co-simulation execution, it is possible to define a collection of scenarios. These scenarios can be thought of as test cases and considered as a sequence of external stimuli to the co-simulation. Each stimulus has a time variable associated with it, such as an event that should occur at specific points in time during the co-simulation run.

One necessary condition for a co-simulation to be consistent is that all its artefacts should not impose any contradictory aspects. This seems a natural condition, but contradictions, especially semantic level contradictions among co-simulation artefacts, are not always easy to identify. Therefore, we propose a Model Management System (MMS) to provide methods to integrate co-simulation artefacts, keep models synchronized and ensure consistency during concurrent model evolution.

**MODEL MANAGEMENT SUPPORT**

This section introduces the architecture of the model management system and tool support to keep co-simulation artefacts organized and synchronized. Several consistency checking schemes are proposed to prevent contradictions between domain models. A co-evolution graph is also introduced to provide a common platform for domain experts to visualize the development process, and ensure consistency during concurrent co-modelling and co-simulation processes.

**System Architecture**

The Model Management System (MMS) is a service-oriented system that provides a set of services to facilitate the organization and manipulation of all objects produced in a co-simulation development, including model resources and development information. The MMS architecture is depicted in Figure 3, which comprises three layers, namely the Model Base Layer, the Application Layer, and the UI Layer. Each layer is in-
modelling information: the Model Metadata

The model base layer mainly contains three kinds of information: the Model Metadata, Relationship Metadata and Model Files. The Model Files represent the physical storage of all modelling related files of one co-simulation development, including variants of DE models, CT models, co-simulation contracts, test scenarios, simulation results, and documentations. Modellers who have the authority are able to access the model files, make changes, and reuse these model elements. This component is implemented as a distributed repository using the JGit library (i.e. a Java library of the Git revision control and source code management system), such that the entire model repository can be kept remotely and locally to prevent data loss.

The Model Metadata contains the meta-information of all Model Files and composite information about them, such as the co-model and co-simulation run, shown as (2) in Figure 4. A meta-data binding algorithm is established to ensure the consistency and synchronization between basic model files and their compositions. The meta-data binding is bidirectional; therefore, if a binding has been set, any changes on either side will reflect each other in order to keep them updated. For example, if a CT-domain expert changed a CT model which exists in a co-model element or a co-simulation run element, both these composite elements will be updated to reflect that change, and vice versa.

Since the DESTECS co-simulation framework allows modellers to explore design alternatives and design space, and also provide reasons for their design decisions, the Relationship Metadata defines a set of model evolution relationships that we collected from previous co-modelling and co-simulation experiences. The main purpose of having these relationships is to capture the concurrent model evolution process and provide decision making support such that inconsistencies between evolving models can be reduced.

Three types of relationships are abstracted: design alternative, design step, and design baseline, shown as (3) in Figure 4. In practice, it is common that more than one candidate design is made for the same development purpose in order to find optimal design solutions. We call these candidate designs design alternatives. Particularly, the design alternative relationship can be constructed between DE models, CT models, co-models and scenarios, respectively. While designs are evolving, the design step relationship represents the steps that move a design towards its final form. This may entail the addition of detail to make a model more competent for a proposed implementation. The design baseline relationship designates a particular design as significant in the development story, where we might expect to see a full set of scenarios, results, rationales, and documentations associated with such baselines. All these relationships are essential for keeping track of the development lifecycle and needed to generate the co-evolution graph.

Application Layer

The Application Layer consists of two submodules: the MMS Services and the Interfaces. The former provides higher level functions for manipulating model files and model metadata stored in the model base, as well as information that can be accessed by the UI Layer. The latter is used to facilitate communication between model management services and the model base layer. Within the MMS services module (see Figure 3), the Model Base Configuration, Model Integration, and Relationship Binding are fundamental services concerning the integrated co-simulation framework. These services provide integration and binding solutions to keep individual artefacts synchronized such that a co-simulation development can be managed as a whole during its entire life-cycle.

The Consistency Checking service provides methods to check the syntactical consistency between two domain models and the co-simulation contract, by means of checking the SharedProperty object, shown as (4) in Figure 4. Model interfaces have to specify the shared properties with the exact data type, identifier name, and access level in order to execute the co-simulation. The consistency checking service creates shared property objects for all DE and CT model interfaces and contracts. For instance, when composing a co-model or a co-simulation run, the consistency checker will examine each shared object for all elements. This service is also used for model searching and navigation purposes. When seeking potential CT domain design alternatives from the model base, the consistency checker traverses the entire model base to find consistent counter-part models which contain the same shared objects.

Figure 3: Model management system architecture
Together with the Versioning Support, the Co-evolution Support keeps records of all model evolution relationships and the design history of the development. With these services, modellers are able to analyse the design decisions made by other domain experts and locate possible inconsistencies that happened in the past, such that engineers can gain more common modelling knowledge and optimize their designs. A graphical visualization can be generated based on the information provided by the co-evolution support, which will be explained in the next subsection.

**UI Layer**

The User Interface Layer provides views and dialogs to facilitate the access of the model base. Consistency, dependency and synchronization issues are taken care of by the MMS services module. Modellers with valid credentials can also access multiple model bases to import and reuse models. Models that are imported from other model bases can be either embedded or linked. The former is a full-copy of the model to the current model base and can be edited locally. The latter is considered as a reference to the model; a linked model can only be read. Any changes made to the referenced model will have an impact on the target model base.

The Co-evolution Graph Viewer is to show the evolution process of all model base elements, from planning and structuring towards developing, design decision making, analysing, reusing, and finally discarding or approving.

The information contributed to the evolution process \((EP)\) comes from the relationship metadata and can be defined as: \(EP(o, \Delta t) = \{H, ALT, DS, B\}\). This means that the evolution process of an object \(o\), within a time period \(\Delta t\), contains the development history \((H)\), the alternatives \((ALT)\), design steps \((DS)\), and baselines \((B)\) that represent the rational and design intents behind these changes. An object can be either a domain model, a co-model or a co-simulation run, where design alternatives are only available for domain models and co-models.

Therefore, an evolution graph can be defined as a set of interconnected objects \(o\), with relationships \(ALT, DS\), or \(B\) over time. The objects are expressed as a collection of coloured Nodes and the interconnections are Edges, where both nodes and edges can have attributes, such as time, author, description, parents, children, etc.

Each model modification point, alternative, design step, and baseline making point is represented as a node. Depending on the type of the node, corresponding attributes are attached. Different types of edge represent different relationships between two nodes. One node can connect to multiple types of edges. For instance, a node \(n\) connects to another node \(m\) with one \(DS\) type edge, and also can connect to nodes \(p\) and \(q\) via two \(ALT\) edges.

An example graph is shown in Figure 5. Model growing over a long period of time may result in a large number of models. Therefore, the graph can be shrunk by showing fewer uninteresting models (tagged by mod-
Figure 5: A conceptual example of the co-evolution graph

ellers as minor during development phase) and replacing the nodes by dotted lines to narrow down the information to domain experts, such that more important and useful information is shown. Modellers may expand the graph to view the entire evolution process. Searching algorithms are available to find relevant models.

EXAMPLE: LINE-FOLLOWING ROBOT

This section describes a robot design using the model management support within its design process. The entire life-cycle of the DE model, CT model and co-model, from creation to end, is tracked; this information is used to generated the (co-)evolution graph. Some design guidelines on how to prevent inconsistencies during co-simulation development are explained as well.

Problem Specification

The goal of the case study is to develop a line-following robot to track lines on the floor. A snapshot of the 3D representation of the robot is shown in Figure 6. The focus of this case study is to design such a system in the DESTECS co-simulation framework, and demonstrate the concurrent design flow using the model management system.

The target robot consists of the following features relevant to our paper: two black/white coded wheels, connected to two servo motors with built-in feedback controllers; an angular position measuring element to obtain the angular position of the wheel; infrared line-following sensors that sense contrast in the range of 0 to 255 bits. The example in Figure 6 is one alternative design of the system; two white spheres represent the two line-following sensors mounted on the front. The sphere turns black when the sensor sees a black line. More details of the design can be found in Broenink et al. (2012) and Pierce et al. (2012).

Figure 6: 3D representation of one design alternative of the robot in 20-sim tool

Iterative Design Steps

Following the iterative modelling process described in Figure 1, the iterative development process of the line-following robot is described in this section, as well as problems found during development. This section also shows the benefits of using the MMS tool and the (co-)evolution graph to help with decision making and reuse of models.

• Phase 1: Requirements and Analysis

Based on the robot features described in previous section, modellers from different domains worked together and wrote a list of assumptions about the robot and environment (size and weight of the robot, sensor ranges, contact surface etc.) before defining an initial co-simulation contract. The list of assumptions is to help dealing with inconsistencies between domain models, because inconsistencies usually happen due to different interpretations of shared properties. Therefore, providing extra semantic information to all shared properties can reduce the chance of leading to inconsistent models. Such semantic information is, for example: data type, measurement unit, range of acceptable values, and the direction of positive values or reference frame.

During this phase, two major design steps are constructed which push the design from a simple robot co-model to a more complex and mature stage: Design Step 1, path following of a path known to the controller (simple, no line-following sensor involved), and Design Step 2, line-following of a path using two line-follow sensors.

• Phase 2: Co-model and Co-simulation Run Creation

In this phase, an initial co-model towards Design Step 1 is composed and added into the model base via MMS, with mappings to each domain model and the contract. The initial points for all DE models, CT models and co-models are kept by the MMS and can be displayed in the (co-)evolution graph as the nodes 1 in Figure 7(a), 7(b) and 7(c), respectively.
Figure 7: Evolution graphs of the line-following robot design, in which the green-border nodes represent design steps, the yellow-border nodes represent alternatives, and the yellow nodes are currently selected models (the blue texts and red arrows are not part of the tool).

Changes to each domain model and contract reflect on the mapped co-model, so the co-model also inherits the design history of its child elements, such that co-evolution is governed. Together with co-simulation settings (e.g. simulation time) and the scenario script (i.e. pre-defined run-time commands), a co-simulation run is then stored in the model base. Similar to the co-model, a co-simulation run composes a set of mappings to its children elements in the model repository. The mapping is bidirectional, so changes to the co-model or co-simulation run will also influence domain models or contracts.

Phase 3: Co-simulation and Design Decision Making

Many design decisions are made after evaluating the co-simulation results in order to improve and refine the co-model. We hereby concluded several important decisions that influence the entire design of the robot. During the design towards Design Step 1, some inconsistency problems occurred. The CT model became buggy when incompatible 20-sim components were added into the robot dynamic model (shown as ② in Figure 7(b)). Meanwhile, the DE modellers tested their DE controller based on the lastest working version of the CT model.
( 2 in Figure 7(a)), and another CT modeller started adding extra functions to the working CT model while bugs were being fixed ( 3 in Figure 7(b) and Figure 7(c)). Due to time constraints, the buggy CT model was finally abandoned ( 4 in Figure 7(b)). From Design Step 1 to 2, both contract and domain models are evolving. Adding extra sensors requires the extension of the co-simulation contract and model interfaces. At this point, the consistency checker checked both the interfaces and the contract to make sure the evolved co-model was still consistent.

A primary goal for the line-following robot is that it does not stray from the line, however the co-simulation result shows large deviations of the actual moving trajectory. Therefore, different design parameters ( 5 in Figure 7(a), and 6 in Figure 7(c)) are tested to find which is a better solution to keep the robot moving curve close to the line. Similar design decisions are made often in this phase of development, such as testing the influence of sensor position on the robot or the sensitivity of sensors and controller to correct the errors etc. So, modellers are recommended to frequently update their models and relationships to the model management system such that this information can be retained.

- Phase 4: Verification and Evolution Graph Evaluation

Based on development activities produced in Phase 2 and Phase 3, three evolution graphs are shown in Figure 7 for the line-following robot development. Each node in the graph indicates a model at certain time. Unimportant versions of models are hidden in the graph and use dotted lines to indicate that two nodes are indirectly connected. Solid lines are direct connections between models or model versions. The evolution graph can be generated and analysed to get an overview of the development process, and help modellers to understand each other and cooperate better.

The co-model evolution graph shown in Figure 7(c) can be considered as the co-evolution view of different types of models. Once domain models are changed, a new co-model “node” will be created to store the information of change. The co-model evolution graph is not simply overlapping the development process on two domain models, but also provides more details on the development decisions of the robot as a whole. Each node indicates which joint decisions are made during development. These evolution graphs show all interesting design decisions and relationships between models. Generating these evolution graphs while co-modelling allows modellers to examine previous design decisions, in order to identify which designs they did wrong and why, which assumptions and decisions made by different domain modellers are not consistent, etc., such that modellers can refine their models in further development. In addition, models become easier to reuse and extend.

CONCLUDING REMARKS

In this paper, we advocate the co-simulation-in-the-loop development process using the DESTECs co-simulation framework, which highly reduces the design time and costs for cyber-physical system designs. With this framework, different domain models can be produced and verified together without implementing any physical prototypes. However, a huge amount of models and model variants may be produced during development which increases the design space and the difficulty in managing it.

Therefore, we proposed a Model Management System (MMS) and its tool support to facilitate the co-simulation development process and prevent inconsistencies. The main contributions of the work are: (1) providing a distributed model repository for collaborative modellers, (2) capturing model dependencies and keeping them synchronized during model co-evolution, (3) supporting model consistency checking when composing co-models and seeking design alternatives from the model base, (4) visualizing concurrent model evolution processes in order to locate possible inconsistencies made in the past, and (5) testing the MMS tool and iterative design flow via a case study in which possible inconsistencies are prevented.

ACKNOWLEDGEMENTS

The authors thank all people from WP2 of DESTECs for sharing insight and experience of co-modelling and co-simulation and Ed Schaefer and Robert Wilterdink from the RaM group for implementation help.

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 (DESTECs Project).

REFERENCES


Giese H.; Hildebrandt S.; and Neumann S., 2010. Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. In Graph Transforma-


