9-1-2006

An Instance-Based Structured Object Oriented Method for Co-Analysis/Co-Design of Concurrent Embedded Systems

Matt Ryan

Xiaoqing Frank Liu
Missouri University of Science and Technology, fliu@mst.edu

Bruce M. McMillin
Missouri University of Science and Technology, ff@mst.edu

Ying Cheng

et. al. For a complete list of authors, see https://scholarsmine.mst.edu/comsci_facwork/178

Recommended Citation

M. Ryan et al., "An Instance-Based Structured Object Oriented Method for Co-Analysis/Co-Design of Concurrent Embedded Systems," *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06) (2006, Chicago, IL)*, Institute of Electrical and Electronics Engineers (IEEE), Sep 2006.
The definitive version is available at https://doi.org/10.1109/COMPSAC.2006.26

AN INSTANCE-BASED STRUCTURED OBJECT ORIENTED METHOD FOR CO-ANALYSIS/CO-DESIGN OF CONCURRENT EMBEDDED SYSTEMS

Matt Ryan, Sule Simsek, Xiaoqing (Frank) Liu, Bruce McMillin
Department of Computer Science
Intelligent Systems Center
University of Missouri – Rolla
{mjr, simsek, fliu, ff}@umr.edu

Ying Cheng
Department of Electrical and Computer Engineering
University of Missouri – Rolla
ycheng@umr.edu

Abstract

The current object-oriented class-based approaches to hardware/software co-analysis/co-design of embedded systems are limited in their abilities to properly capture the structure of individual instances of hardware and software components and their interactions. This paper discusses a methodology to extend a structured object-oriented hardware/software co-design methodology based on the High Order Object-oriented Modeling Technique (HOOMT) to incorporate instance-based object and behavioral models. The instance-based structured object-oriented methodology will enable description of a system's structure based on individual instances of hardware and software components and specification of the interactions among them. In addition, lattices are introduced to specify the concurrent behavior of hardware/software components in a concurrent embedded system. These additions further enhance the method's capability of providing a precise set of specifications that can be understood easily by both hardware and software designers in co-analysis/co-design. The methodology has been applied to the specification of hardware and software of a simulated advanced power grid control system.

Keywords

Hardware/Software Co-analysis, Hardware/Software Co-design, Instance-Based Co-Analysis/Co-Design, Structured Object-Oriented Method, Concurrent Process, Integration, Embedded Systems

Supported in part by the UMR Intelligent Systems Center and supported in part by NSF MRI grant CNS-0420869.

1. Introduction

The current crop of object-oriented methods (such as [1], [2], and [3]) for the co-analysis and co-design of hardware and software components of embedded systems is primarily class-based, with each of the system components represented as a class at the higher levels of design. This class-based bias is derived from the traditional object-oriented software design approaches which emphasize object classes for the general design. By contrast, hardware design is usually done using what can be termed instance-based methods, involving such tools as circuit and wiring diagrams, system architecture diagrams, etc., such as the designs shown in [4]. It is necessary that the hardware design indicate the specific number of instances of a component, and how those instances interact with each other and the other system components.

Class-based approaches to object-oriented co-analysis and co-design present two key problems to embedded systems design. First, while class-based diagrams capture a significant portion of a system's structure, they often fail to properly capture or express system structure and behavior where multiple instances of the same component class are involved. The second problem with class-based object-oriented methods stems from the different backgrounds of the hardware and software designers. While software designers are often accustomed to creating and viewing designs that contain object classes and their relationships, hardware designers are much more used to looking at system diagrams in which all instances of the components and their relationships and interactions are laid out. Instance-based object-oriented co-analysis and co-design methodologies attempt to address these two shortcomings, allowing for enhanced specification of a system's components, in addition to providing system specifications that are much more easily

Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)
0-7695-2655-1/06 $20.00 © 2006 IEEE

understood by the hardware designers as well as the software designers.

1.1. Structured Object-Oriented Co-analysis/Co-design of Hardware/Software Using HOOMT

Previous work by one of the coauthors of this paper developed a structured object-oriented software design methodology based on hierarchical development called the High Order Object-oriented Modeling Technique (HOOMT) [6]. The integration of structured methods with object-oriented methods provides the uniformity and reusability of the object-oriented approach with the hierarchical decomposition of objects, their functions, and their dynamic behaviors that is provided by the structured method.

This methodology was developed further by the coauthors and incorporated into a method for the co-analysis and co-design of hardware and software of embedded systems [7]. This HOOMT method provides a systematic approach that guides the co-analysis/co-design and the natural partitioning of the design into its hardware and software components. Our current research proposes an extension to the hardware/software co-analysis and co-design methodology to include the development of instance-based object and behavioral models to further explore the structure and interactions of hardware and software components of embedded systems.

1.2. Distributive Lattices for Dynamic Behavior Specification

In order to explore the interactions and concurrent behavior of the components of embedded systems, we borrow the concept of distributive lattices [11] from distributed systems. The components (objects) in the embedded systems interact by message passing. Each message corresponds to an event in the embedded system. The terminology regarding distributive lattices and the timing constraints on interactions in embedded systems is provided below:

Definition 1 Spatial constraints on the events in embedded systems: If the event is an interaction between different objects it is called a global interaction (event).

Definition 2 Temporal constraints on the events in embedded systems:

(1) If **a** and **b** are events on the same object, and **a** comes before **b**, then **a** → **b**; and **a** & **b** are sequentially related.

(2) If **a** is the sending of a message by one object, and **b** is the receipt of the same message by another object, then **a** → **b**; and **a** & **b** are causally related.

(3) Two distinct events, **a** and **b**, are said to be concurrent if ¬ (**a**→**b**) and ¬ (**b**→**a**) [10].

Definition 3 Consistent Global State: The entire system state, which is a collection of concurrent events, consisting of the states of all objects based on their interactions.

Definition 4 Distributive Lattice Representation: The collection of all possible consistent global states of interactions observed in an embedded system.

Definition 5 Vector Time Stamp: A vector of discrete clock values, one from each object in the distributed embedded system [12].

Definition 6 Lattice: A lattice L is a partial ordered set P such that for all x, y $\in P$, there exists a greatest lower bound and a lowest upper bound.

Definition 7 Distributive Lattice: A lattice L is said to be distributive if the distributivity property holds on all x, y, z $\in L$.

The components of embedded systems can interact concurrently as well as sequentially. Although representation models such as UML sequence diagrams are capable of representing the sequential interactions, they are not capable of properly representing concurrent interactions. Therefore, it is necessary to have a model capable of representing both sequential and concurrent interactions between objects. In order to observe the concurrent interactions of the embedded system components, we utilize the distributive lattices. Figure 1 shows three sequence diagrams which represent the interaction of three objects. It is important to note that these three sequence diagrams represent exactly the same interactions between objects; however, traditionally they are treated as different interactions. Here we show that these three sequence diagrams are represented with the same distributive lattice, therefore, clarifying any misrepresentation issues.

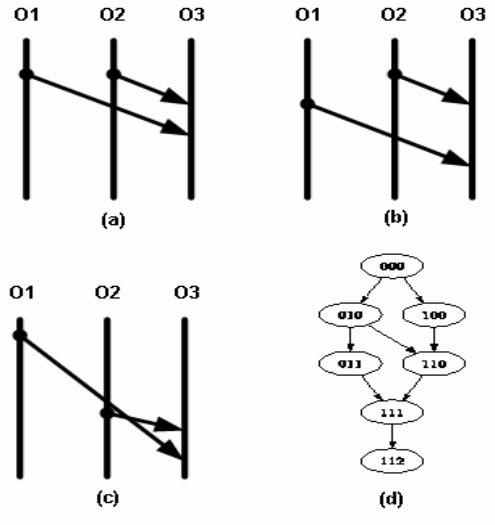


Figure 1: (a) (b) (c) Three sequence diagrams represent the same interaction of O2 sending a message to O3, followed by O1 sending a message to O3. (d) All three sequence diagrams correspond to the same distributive lattice.

The vertices of the distributive lattice represent the consistent global states, composed of vector time stamps of all the objects in the system. Since there are three interacting objects, three digits are used to represent each object's state. The vertices at the same level represent concurrent consistent global states of the embedded system, whereas the vertices at different levels which are reachable from each other represent sequential consistent global states of the embedded system. The edges of the lattice represent the global events observed in the embedded system. By going through a lattice path, sequential, causal, and concurrent interactions of objects in embedded systems can be observed.

The distributive lattice representation model of object interactions is able to capture the concurrent interactions of instance-based as well as class-based objects in embedded systems. The observations obtained from the distributive lattice model will be integrated into the dynamic behavior specifications for a more complete view of system behavior.

1.3. Advanced Power Grid Control and the FACTS Testbed Simulation

Power network control has become an extraordinarily difficult task due to the sheer size of such networks. It is desirous to attempt to mitigate the effects of single contingencies (such as line failures) as they occur, before some combination of contingencies can lead to a cascading failure scenario in which most or all of a power grid goes down.

The family of "Flexible Alternating Current Transmission System" (FACTS) devices shows promise for use as network-embedded controllers [8, 9]. There is ongoing research to incorporate a number of FACTS devices that contain embedded communicating distributed computers into a power grid network to act as a distributed, fault-tolerant, and real-time constrained control system. This paper looks at the integrated, instance-based structured object-oriented co-analysis/co-design of a FACTS-augmented power system, specifically a hardware-in-the-loop test system that is currently being implemented to test FACTS control of a simulated power system. This test system includes a multiprocessor simulation engine that will use mathematical formulae for simulating a power grid, and send appropriate power generation commands to three actual power lines, which will have FACTS devices attached to them.

The remainder of this paper is organized as follows. Section 2 discusses the instance-based extensions to the HOOMT co-analysis and co-design process. Section 3 presents a number of instance-based High Order Object Model diagrams from the modeled system which were generated based on transformations from their class-based diagrams, and Section 4 presents samples of distributive lattices to model the system's behavior. Section 5 discusses the results of the modeling effort. Section 6 contains the conclusion to the paper.

2. The Instance-based HOOMT Process for System Co-analysis and Co-design

The original class-based structured object-oriented HOOMT method for performing the co-analysis/co-design phase of embedded systems development provides a unified method for the specification of a target embedded system, including both hardware and software components. The method allows the partitioning of the hardware and software components to be performed, and the interfaces between components can be specified. During the succeeding phases of concurrent component implementation and component integration and testing, continued communication between the hardware and software designers allows for continued refinements to the system components, as well as the possibility for the migration of components between hardware and software as needed.

The structured object-oriented hardware/software co-analysis/co-design phase itself consists of two stages: system level co-analysis and co-design, and component level co-analysis/co-design. The hardware and software designers start with the system level co-analysis/co-design and working together create the HOOMT models of the whole system. Once these models have been sufficiently hierarchically decomposed to show primitive objects, an initial partitioning of the system into the hardware and software components is performed, and each group of designers takes their respective specifications for further development in the component level co-analysis and co-design stage.

Figure 2(a) shows the proposed extension to the HOOMT co-analysis/co-design method. This extension takes place in conjunction with the structured object-oriented co-analysis/co-design phase. Prior to component partitioning, the class-based models are transformed into instance-based models to further specify the component relationships and interactions. From the instance-based models, further refinement into more detailed physical (implementation-specific) component specifications can take place to supplement the component level co-analysis/co-design.

Figure 2(b) shows the transformations of the *Bus* and *Power Line* object classes (such as would be found inside a *Power Transmission System* object). The objects are first instantiated in an instance-based object model, here shown as an individual power line instance connected to two separate bus instances. Also at the instance level, distributive lattices can be constructed to help show component interactions and behavior which can't be

captured at the class level. Additionally, further models can be added at the instance level to simplify specification in cases where a complete instance-based object model would be too cumbersome to fully represent the instantiated components. Using the bus/power line example, a digraph representing the individual buses and power lines could be created to show which particular buses are connected by which power lines for a given system.

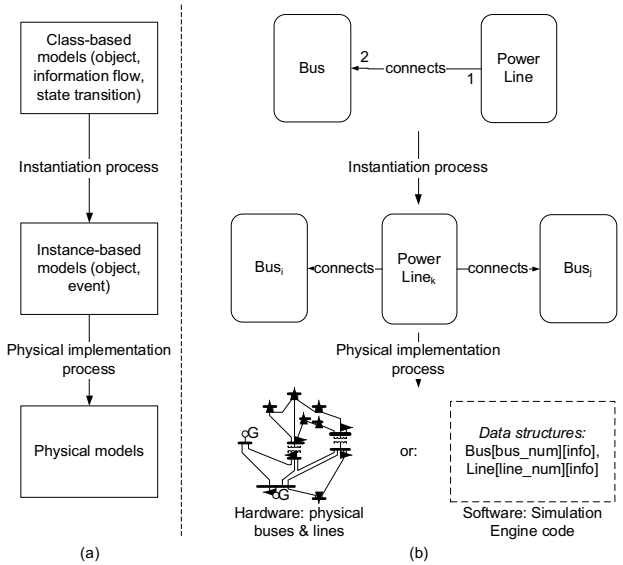


Figure 2: (a) The extended HOOMT development process. (b) Example of bus/line specifications using the development process.

From the instance-based models, the specifications of the bus and power line component instances can be taken to the specific designs required for implementation. The figure shows two possibilities for the implementation of the buses and lines, as either hardware, in the form of wiring diagrams showing the layout and connections of the components in an actual power transmission system, or as software, in the form of a set of data structures (specifically a bus and a line matrix) to be used in code for the dynamic simulation of a power transmission system.

2.1. Development of Instance-Based Object Models Based on Instantiation from Class-Based Models

The development of the instance-based models begins with the transformation of the High Order Object Model (HOOM) created during the system level co-analysis/co-design stage of the HOOMT process. Beginning with the top level object model, the component objects are instantiated. This process starts by adding instances of components to the diagram as needed. Once the requisite components are in the diagram, the attributes and constraints of each instance can be individualized appropriately. Next, the relationships between objects need to be updated to include the new object instances in the diagram. Once the diagram has been instantiated, the process continues at subsequent levels of decomposition of the complex components of the system, following the original object decompositions provided by the class model. The decomposition can continue until either all objects in the diagrams are primitives, or until there is no further need to show the instantiation of components at lower levels.

Once the instance-based diagrams are created, it may be desired to create a complete system structure model, in which the component instances at all levels of abstraction are combined into one diagram. The system structure diagram provides a single diagram to show the entire instance-based structure of the system at all levels, helping to further understanding of the system specification by both the hardware and software designers.

2.2. Distributive Lattices for Modeling Instance-Based Behavior

The basic system structure is captured by using the instance-based object models. In order to represent the concurrent interactions of the objects in instance-based object models, distributive lattices are used. These distributive lattices are re-termed “instance lattices” to distinguish them from lattices used for class-based object models. The advantage of using these instance lattices is in revealing the concurrent interactions between components at the instance-level which are not visible at the class level. In addition, the lattices can be created in a hierarchical manner following the decomposition of the object models in order to show the decomposition of the higher-level interactions present at the previous abstraction level.

For a particular scenario of component interactions, it is first necessary to create the vector time stamps of the interactions taking place. Then a lattice is created to show all possible consistent global states that may occur over the course of the scenario. Such scenarios may be generated from existing diagrams in the class-based Hierarchical State Transition Model (HSTM), combinations of HSTM diagrams with information from the Hierarchical Object Information Flow Model (HOIFM), or may be brand-new scenarios developed by the designers based on knowledge/assumptions of what the embedded system components should be doing. Lattices can be created for multiple levels of abstraction as needed, for as many scenarios as desired. The generation of vector time stamps and the distributive

lattice may be done by automated programs/tools, such as the LatGenU program [5].

3. Instance-Based Transformation of the HOOM for Hardware/Software Co-analysis/Co-design

The transformation of the HOOM into a set of instance-based diagrams, as well as an integrated system structure model, begins at the top level of the class-based model created during the initial co-analysis/co-design process. (If needed, the actual upper-most level, or context object diagram, can be skipped in the event that an instance-based version of the diagram looks identical with its class-based template.) Figure 3 shows the top level class model for the simulated power testbed system currently being designed and constructed as part of the research effort.

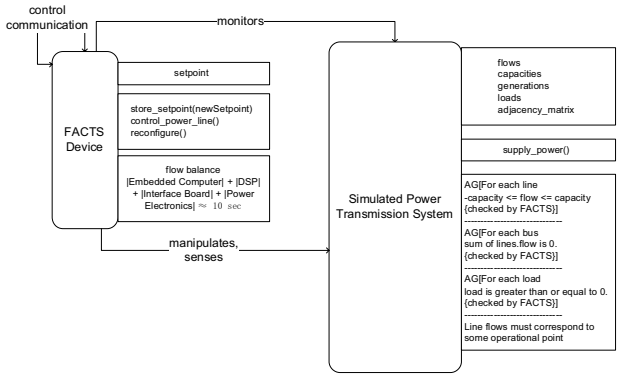


Figure 3: Class-based top level object model.

In Figure 3, two components are visible: the *FACTS Device* and the *Simulated Power Transmission System*. (For simplicity, some of the components in the original model, such as the *Placement* and *Contingency* object classes, have been abstracted out.) Also shown are the relationships between the three components, including a relationship labeled "control communication" that appears to be between the FACTS Device and itself. This relationship, originally intended to indicate the communications between multiple FACTS devices on a power grid, is a prime example of the limitations of class-based models, as will be seen when the relationship is clarified upon instantiation.

At the current level of abstraction, the FACTS Device class object is instantiated into three FACTS Device instances. Each instance is given a number from 1 to 3 to distinguish it from the others. The relationships between the three FACTS devices, as well as the relationships between each of the FACTS devices and the Simulated Power Transmission System (SPTS), are next instantiated. These relationships are carefully instantiated so that their original cardinalities carry over into the instance-based model. For example, the "manipulates, senses" relationship, which represents the interaction of each FACTS device with its HIL line, is a 1-1 relationship, and thus each enters the SPTS at a different point. The "monitors" relationship, however, represents the sensor data of the whole power transmission system that would be sent out to all of the FACTS devices, hence a one-to-many relationship, and is instantiated as such. Similarly, the original "control communication" relationship has been replaced by a one-to-many relationship that connects all three FACTS devices. The resulting model can be seen in Figure 4.

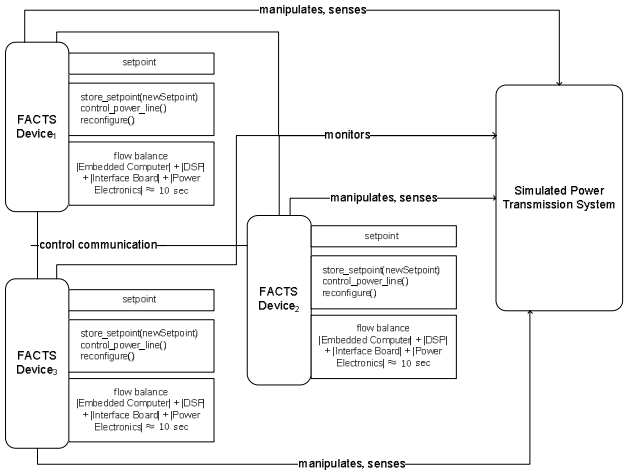


Figure 4: Instance-based top level object model.

In addition to the objects and their relationships, as they become known, the individual attributes of the components can be instantiated and added to the model. For the FACTS devices, this would include such attributes as what line in the simulated system each was "attached" to. Similarly, the constraints on the components can be instantiated, or "individualized", as they become known.

After instantiating the FACTS devices in the top level diagram, the transformation of the class-based model continues with the high order *Simulated Power Transmission System* object. Looking at the decomposed class model reveals that the SPTS contains two object classes, the *Simulation Engine* and the *HIL Line*, and their respective relationships with each other and with objects outside of the SPTS (the FACTS Device and Contingency). To create the instance-based object model of the SPTS, there must be three HIL lines and one Simulation Engine. Figure 5 shows both the original class-based model and the resulting instance-based diagram. (Due to space constraints, the models presented here have been simplified by removal of the attributes, methods, and constraints.)

The Simulation Engine component's instantiated attributes include the configuration information (buses, lines, generators, loads) of the power transmission system to be simulated. As discussed above in Section 2, once the model has been further transformed into the physical (implementation) model, the bus and line objects will manifest as both a set of data structures in the Simulation Engine, the bus and line matrices, and as a physical set of buses and lines, contained within the HIL line objects, that will duplicate the power flows of three chosen power lines in the simulated system.

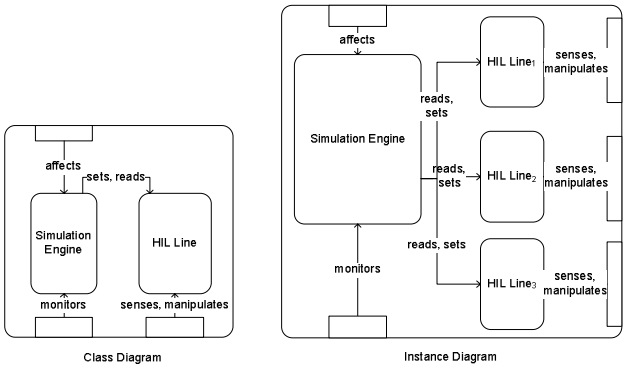


Figure 5: Class and instance-based Simulated Power Transmission System models

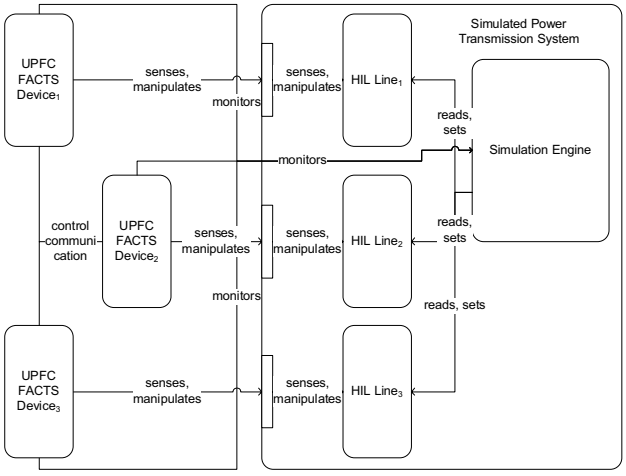


Figure 6: System structure diagram.

At this point in the process, further instantiation of lower levels in the model's hierarchical decomposition is stopped to prevent the integrated system structure diagram from becoming unmanageable. Further instantiations at lower levels of abstraction can be continued as needed for further exploration of component structure and interactions. To create the system structure diagram, the instance-based decomposition of the Simulated Power Transmission System is incorporated into the instance-based top level diagram in Figure 4. The relationships between each of the three FACTS devices and their corresponding HIL lines are matched up, with the ports at the object edges maintaining the proper separation for relationships that cross object boundaries. In addition to the incorporation of the instanced-based SPTS components, each of the three FACTS devices is re-instantiated as a specific type of FACTS device, a Unified Power Flow Controller, or UPFC, FACTS Device. The completed system structure diagram for our model is presented in Figure 6.

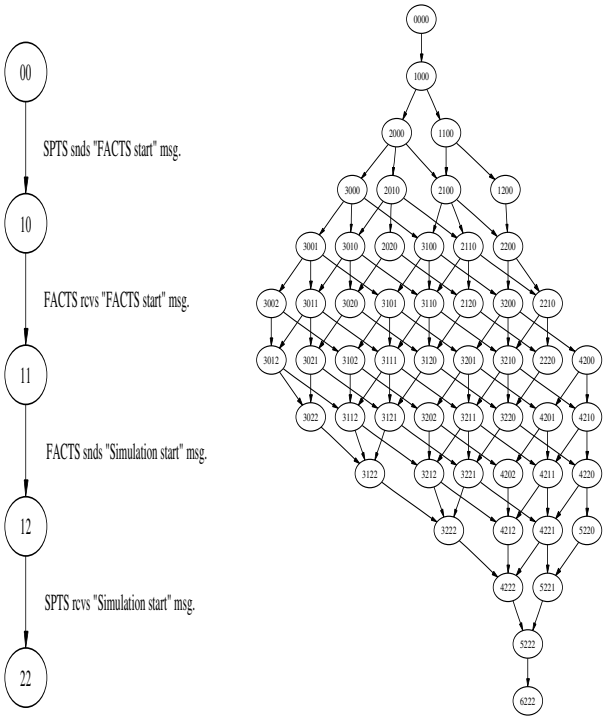
From the system structure diagram, the designers can see the whole instance-based structure of the system at one time, and use it for further design work, including the creation of the instance lattices for modeling scenarios of events.

4. Class- and Instance-Based Behavior Modeling of the FACTS Power System using Distributive Lattices

For the instance-based co-analysis/co-design process, distributive lattices are one of the methods being added to the HOOMT for use in capturing the dynamic behavior of a system. These lattices are a means of visualizing behavior scenarios. One of the scenarios created for the simulated testbed system under development involves the component interactions that take place during simulation startup. These interactions take place between the Simulated Power Transmission System (consisting of the Simulation Engine and the HIL Lines), and the FACTS Device objects. For this scenario, the class- and instance-based object models were used to identify the components involved, and the global interactions of the components during the scenario listed. At the top level, using a class-based lattice (corresponding with Figure 3), the interactions between the FACTS Device and the SPTS are viewed as a total order sequence without any concurrencies. These interactions consist of messages sent and received by the two objects. In Figure 7(a), the distributive lattice representation of these interactions is illustrated. The vector time stamps of the consistent global states are used to label the nodes of the lattice, which reveal the sequential ordering of the interactions. For this lattice, the first and second transitions represent the sending and receiving of a message from the SPTS to the FACTS Device saying that the SPTS is ready for the FACTS to be started. The third and fourth transitions correspond to the sending and receiving of a message from the FACTS Device back to the SPTS that informs the SPTS that the FACTS Device is ready for the dynamic simulation to begin.

Figure 7(b) shows the top-level instance-based interactions between the single instance of the SPTS and the three instantiated FACTS objects. The concurrent

interactions which cannot be observed at the class-level can now be seen with the instance-level view, as the concurrent consistent global states residing at the same levels appear. It is easy to see that at the class-level all the interactions are sequential; one state follows the next. However, at the instance-level, some states are concurrent and some are sequential (observed through lattice paths). The concurrencies are introduced by the messages sent from the SPTS to multiple FACTS instances and vice versa. By modeling the instance-based components' interactions by distributive lattices, the concurrencies in the scenario which cannot be observed at the class-level are revealed.



(a) (b)

Figure 7: Top level instance lattices (with vector time stamp-labeled states): (a) Class-based component interactions. (b) Instance-based interactions

To examine the component interactions of the scenario closer, the top-level instance-based interactions shown in Figure 7(b) are decomposed one level down, corresponding with the system structure diagram in Figure 6, and the resulting distributive lattice representation is depicted in Figure 8. The new lattice includes the sequential and concurrent interactions between the Simulation Engine, three FACTS devices, and three HIL line objects. The expansion of the lattice represents the decomposition of the high level SPTS object into the Simulation Engine and three HIL Line instances, and the additional interactions between those instances that become visible at the new abstraction level. Similar to the lattice in Figure 7(b), this lattice shows an increase in concurrency up to approximately the halfway point of the lattice, after which the possible consistent global states begin to converge at the last two sequential nodes at the bottom.

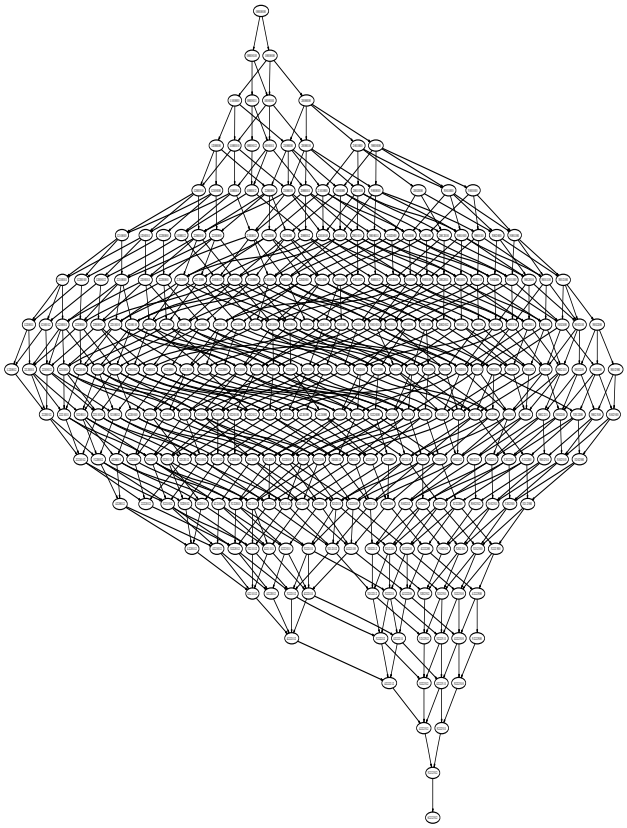


Figure 8: Instance-based component interactions at the second level.

It is important to note that Figure 8 illustrates the interactions at a lower-level than Figure 7. The decomposition enables the observation of more comprehensive interactions between the components. The distributive lattice representation of the lower-level instance-based behavior of the FACTS system reveals a more detailed and complete picture of what can happen during this scenario. The number of paths, sequential and concurrent nodes increases as the model's hierarchical decomposition is further instantiated. It should also be noted that the concurrency, represented as an expansion in the lattice, becomes larger as the lattice is decomposed into lower levels. This same observation of the expansion of the lattice is made in the comparison of the class-based lattice modeling with instance-based. Additionally, instance-based modeling reveals concurrent interactions between the components of the system which are not

visible with the class-based modeling. The observation of concurrent interactions of system components is made possible through the incorporation of the distributive lattices into the instance-based co-analysis/co-design process.

5. Development Results

The instance-based HOOM diagrams and distributive lattices presented in this paper are a portion of the instance-based system specifications of the proposed simulated FACTS testbed system that were developed using the methodology discussed in Section 2. Along with the addition of an instance-based methodology to structured object-oriented co-analysis/co-design, of particular interest was the new application of the distributive lattice concept to hierarchically decomposed instance-based dynamic behavior specification of concurrent embedded systems. The addition of the instance-based object and behavior models to the methodology has proved very useful in the system specifications being produced, providing more accurate representations of the components and their interactions than were present before.

One observation gained from the use of the distributive lattices was the tendency for exponential state explosion with the addition of more interactions (such as during the hierarchical decomposition). It should be noted that the cause of this explosion, the additional possible consistent global states due to the additional concurrent interactions of the scenario, is not a fault of the lattices, but rather of the concurrent behavior itself. Other techniques for representing behavior and interactions would suffer either from similar state-space explosions, or already be limited in their abilities to capture concurrent behavior (such as the Hierarchical State Transition Model already provided in the HOOMT).

6. Conclusions

The use of object-oriented methodologies for hardware/software co-analysis and co-design has been very valuable for the specification of embedded systems. However, existing class-based approaches present difficulties in properly capturing the structure, relationships, and behaviors of component instances, in addition to not always being easily understood by hardware designers. The addition of an instance-based approach to the High Order Object-oriented Modeling Technique allows for an extension of a structured object-oriented methodology for the integrated co-analysis and co-design of individual instances of the hardware and software components of embedded systems. As shown in the above examples, the instance-based extensions to the methodology and application of distributive lattices allow for more accurate specification of embedded system components and their concurrent behaviors. Additionally, the instance-based approach provides models that by looking more like traditional hardware specifications are more easily understood by hardware designers, and are easier to further develop into implementation-specific system specifications.

7. References

[1] P. Green, D. Morris, and G. Evans. “Software technology for embedded systems”. Software Technology and Engineering Practice, 1997. *Proceedings of the Eighth IEEE International Workshop on incorporating Computer Aided Software Engineering*, pp. 402-410, 14-18 July 1997.

[2] O. Rashid, N. L. Passos, and R. H. Halverson. “An Object Oriented Hardware/Software Co-design Paradigm.” *Proceedings of the ISCA 13th International Conference - Computers and their Applications*, pp. 440-443, March 1997.

[3] T. Y. Lee, P. A. Hsiung, and S. J. Chen. “DESC: A Hardware-Software Codesign Methodology for Distributed Embedded Systems.” *IEICE Transactions on Information and Systems*, IEICE Publishers, Volume E84-D, Number 3, pp. 326-339, March 2001.

[4] L. Dong, M. L. Crow, Z. Yang, C. Shen, L. Zhang. “A Reconfigurable FACTS System for University Laboratories.” *IEEE Transactions on Power Systems*, 19(1):120-128, February, 2004.

[5] LatGenU program - Lattice Generator for Unix. Power Research Group, University of Missouri-Rolla, 2004. http://filpower.umr.edu.

[6] X. F. Liu, H. Lin, and L. Dong. “High Order Object-oriented Modeling Technique for Structured Object Oriented Analysis.” *International Journal of Computer and Information Science (IJCIS)*, 2(2):74-96, June 2001.

[7] M. Ryan, S. Markose, X. F. Liu, B. McMillin, Y. Cheng. “Structured Object-Oriented Co-Analysis/Co-Design of Hardware/Software for the FACTS Power System.” *Proceedings of the 29th Annual International Computer Software and Applications Conference*, pp. 396-402, Edinburgh, Scotland, July 2005.

[8] M. D. Ilic. “Fundamental engineering problems and opportunities in operating power transmission grids of the future” *Int'l Journal of Electrical Power & Energy Systems*, 17(3):207-214, June 1995.

[9] B. McMillin, M. L. Crow. “Fault Tolerance and Security for Power Transmission System Configuration with FACTS Devices,” *Proceedings of the 32rd Annual North American Power Symposium*, vol. 1, pp. 5.1-5.9, Waterloo, Ontario, October 2000.

[10] Lamport, L., “Time, clocks and ordering of events in distributed systems,” *Communications of the ACM* 21, 7, July 1978, 558-565.

[11] Mattern, F., “Virtual Time and Global States of Distributed Systems,” “Parallel and Distributed Algorithms: proceedings of the International Workshop on Parallel & Distributed Algorithms”, Elsevier Science Publishers B. V.,1989, 215-226.

[12] Fidge, Colin J., “Timestamps in message-passing systems that preserve the partial ordering,” *Australian Computer Science Communication* 10, 1, February 1988, 55-66.