26 Sep 2005

Structured Object-Oriented Co-Analysis/Co-Design of Hardware/ Software for the FACTS Powers System

Matt Ryan

Sojan Markose

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

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

STRUCTURED OBJECT-ORIENTED CO-ANALYSIS/CO-DESIGN OF HARDWARE/SOFTWARE FOR THE FACTS POWER SYSTEM

Matt Ryan[*†], Sojan Markose, Xiaoqing (Frank) Liu[†‡], Bruce McMillin[†‡]
Department of Computer Science
Intelligent Systems Center
University of Missouri – Rolla
{mjr, smy78, fliu, ff}@umr.edu

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

Abstract

There are several approaches to the hardware/software design in embedded systems, ranging from the traditional sequential methods which focus on the determination of the hardware architecture prior to software design, to newer object-oriented approaches that attempt to apply software engineering methods to hardware design without a systematic process. This paper discusses a structured object-oriented methodology for the integrated co-analysis and co-design of hardware/software systems using an extended High Order Object-oriented Modeling Technique (HOOMT). This methodology offers a uniform method for hardware and software developers to jointly develop the specifications for and partitioning of the hardware and software components of a system, as well as developing the interfaces between components, and allows easy design migration of components between hardware and software. In this paper it is applied to the co-analysis/co-design of the hardware and software of a simulated advanced power grid control system.

Keywords

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

1. Introduction

The conventional hardware/software design approach is a sequential process that traditionally consists of gathering the requirements for the proposed system, determining and developing the hardware and architecture of the system, and then developing the software that is intended to run on the system. However, this sequential development paradigm is proving to be inefficient for embedded systems that rely heavily on their software components. The design of the architecture prior to software development can put harmful restrictions on the software being developed [7]. In addition, a lack of coordinated interaction between the hardware designers and the software designers can lead to additional problems in the integration and testing of the system. Another possible issue lies with the partitioning of the system into hardware and software components. An early (and fixed) partitioning may not provide the most efficient division of functionality between hardware and software.

In an effort to improve the development of hardware/software systems, a number of concurrent design, or co-design, methodologies have been proposed. These methodologies typically involve the specification of a target system to some level, at which point the hardware/software partitioning is performed, and further design and development of the hardware and software components takes place concurrently, with some amount of communication between the hardware and software designers. Of particular interest in recent years has been the introduction of object-oriented paradigms into the co-design field. Object-oriented design methods can be very useful in hardware/software co-design by providing a uniform method for hardware/software system specification. A uniform modeling method can provide developers with increased understanding of both the hardware and software components. Object-oriented methods also provide two other strengths: they allow for component reuse, and they focus on data relationships that are important in the development of large systems. Finally, the use of object-oriented methods can allow for greater flexibility in deciding when and how to do the partitioning of the components, as well as any potential re-partitionings that may be necessary.

[*] Supported in part by NSF IGERT grant DGE-9972752.
[†] Supported in part by the UMR Intelligent Systems Center.
[‡] Supported in part by NSF MRI grant CNS-0420869.

1.1. Related Work

Green, Morris, and Evans [1] have proposed a MOOSE (Model-based object-oriented system engineering) approach in which a behavioral model of the system is constructed and then transformed into implementations for both hardware and software through several intermediate levels, and then they are synthesized. This research effort recognizes the need to delay the partition of system into hardware and software components to allow for the examination of the overall functional behavior of the complete system. However, the partitioning is done only after the multiple transformations are made on the models throughout the process.

In [2], Machado, Fernandes, and Santos present a three level co-design approach to the development of real-time applications which allows the implementation of industrial control-based information systems. Even though this research provides an interesting method for the object-oriented co-design of real time applications, it does not specify when or how to partition the system into hardware and software components.

Rashid, Passos, and Halverson [3] propose a new object-oriented hardware/software co-design method called SHOOT (Software Hardware Object-Oriented Technique). They specify three different types of objects in this method: hardware, software, and optimizers. It can be easily extended towards any scheduling technique. However, this methodology does not address the issues regarding separate treatment of hardware and software components and their final integration.

A hardware/software co-design methodology for distributed embedded systems called DESC (Distributed Embedded System Codesign) was proposed in [4] by Lee, Hsiung, and Chen. It introduces a two level partitioning technique: (1) design exploration to determine the number of processors for software execution and the hardware cost; and (2) hardware/software co-partitioning to produce a final system partitioning result. Although this research addresses the issues involved in the co-design of distributed embedded systems, like the MOOSE approach it requires the generation of several different models before the system can be partitioned into hardware and software.

Previous work by one of the coauthors of this paper developed the original High Order Object-oriented Modeling Technique (HOOMT) [6]. The HOOMT provides a structured object-oriented software design methodology which is based on hierarchical model development. 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.

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

In the development of embedded systems the importance of the concurrent analysis (or co-analysis) is often neglected, and emphasis is given to co-design and implementation [5]. The importance of the analysis phase lies in the fact that errors introduced in this phase are more expensive to fix later on, especially in embedded systems where both hardware and software components are involved. Therefore, an effective analysis process which can cater to both hardware and software is necessary early in embedded systems development. Our research proposes a co-analysis and co-design process wherein the system requirements are analyzed by a joint team of hardware and software engineers. The co-analysis process gives equal importance to both hardware and software aspects of the system and prevents the design from becoming more hardware oriented or software oriented.

The HOOMT was originally developed for the design of software systems. It is extended to the hardware/ software co-analysis and co-design of embedded systems with a little modification. This modification includes the addition of constraints and the "port" concept to the model (see Section 3 below). The HOOMT provides a systematic approach that guides the co-analysis/co-design and the partitioning of the design into its hardware and software components. Unlike the other methods listed in the previous section, this partitioning occurs in a very natural fashion once the HOOMT models are created.

1.3. Background of the Advanced Power Grid Control System

Power network control has become an extraordinarily difficult task due to the sheer size of such networks. Indeed, as society has become more technology (and thus power) oriented, and the size of the bulk power system grid has increased, the importance of control has likewise grown [8]. The need for better controls has been shown many times, most visibly and spectacularly during the Summer 2003 blackout in the North-eastern United States. In order to prevent similar occurrences, 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 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 into a power grid network to act as a distributed, fault-tolerant, and real-time constrained control system. This paper looks at the integrated, 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 actual power lines, which will have FACTS devices attached to them.

The remainder of this paper is organized as follows. Section 2 discusses the HOOMT process for co-analysis and co-design. Section 3 presents a number of High Order Object Model diagrams from the model system. Section 4 presents a diagram from the Hierarchical Object Information Flow Model of the system. Section 5 provides an example diagram from the system's Hierarchical State Transition Model. Section 6 discusses the results of the modeling effort. Section 7 contains the conclusion to the paper.

2. The HOOMT Process for Co-analysis and Co-design

A HOOMT-based structured object-oriented method for performing the co-analysis/co-design phase of embedded systems development is shown in Figure 1. It provides a unified method for the specification of the target system, including both hardware and software components. The partitioning of the hardware and software components can be performed, and the interfaces between components can be specified using the method. Communication between hardware and software engineers allows for component refinement and migration between hardware and software. The produced specifications can then be used for the later stages of the development process.

During the implementation phase, communication between the hardware and software engineers continues so that new developments that affect the system design can be examined. These developments may lead to further rounds of co-design refinements and potential component repartitioning. Upon creation of the hardware and software components, traditional integration and testing methods can be used to complete the system, allowing for refinements to be made to the components (or the design and specifications) as needed.

2.1. System Level Co-Analysis and Co-Design

The development process begins with the co-analysis and co-design of the system by a joint group of hardware and software engineers using the HOOMT. The first step in the creation of the HOOMT models is the gathering of the system requirements (top of Figure 2). Once the requirements have been identified, a context object diagram of the system is created, showing the system as a

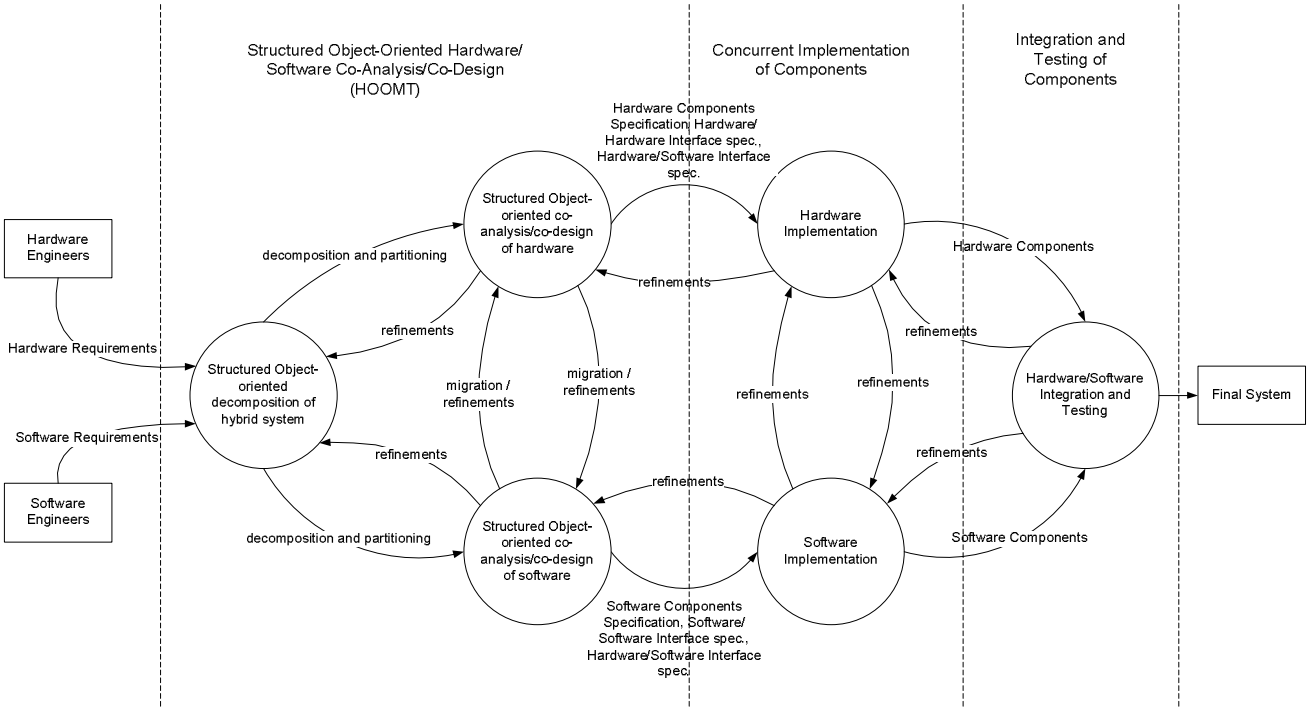

Figure 1: The development process

single entity, as well as the external objects that the system will interface with. The system object is then structurally decomposed one level of abstraction down. All high-order objects identified at the new abstraction level must themselves be decomposed until a level of abstraction is reached where the diagram consists of primitive objects from either the hardware or the software perspective. Additionally, refinements can be made to the higher levels of abstraction in order to maintain consistency. The three system level models of the system are produced, with the lowest abstraction levels of the object model showing the primitive objects, ready to be partitioned into hardware or software components. During the structured decomposition, the object-oriented nature of the methodology allows developers to create both hardware and software views of system objects. These differing views might represent simulated or test-bed versions of the components (versus actual deployed versions), or may allow for an examination of the tradeoffs involved in implementing the component in either hardware or software.

2.2. Component Co-Analysis/Co-Design and Migration

Once the system-level analysis is finished, an initial partitioning into hardware and software components can be performed (bottom of Figure 2). The hardware engineers can take the specifications of the hardware components and their associated interfaces, and proceed with further lower-level component design of the hardware in the embedded system. Likewise, the software engineers will be able to proceed with the continued design of the embedded system software. Continued interaction between the hardware and software engineers is required during these design steps so that changes in hardware or software can be examined for their cross-impact on the components and on the model.

Specifications changes may lead to refinements in either or both the hardware and software component specifications, or may lead to the migration of some system components between hardware and software. The ability to migrate components back and forth between hardware and software during the design phase is one of the major advantages of the structured, object-oriented design process. Such migrations, made simpler by the object-oriented nature of the models, allow for greater flexibility in embedded systems design, particularly in exploring the most efficient division between hardware and software components prior to any actual implementation. The final specifications of the components and their interfaces produced by the co-design step can then be taken for component implementation.

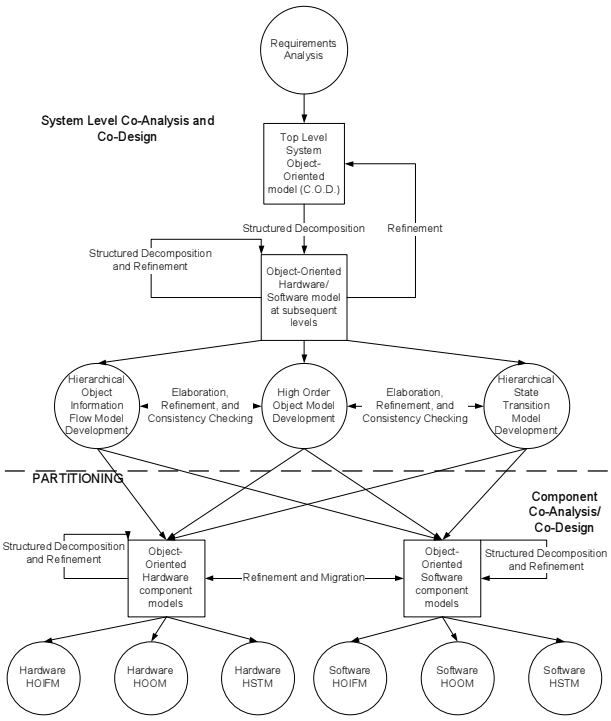

Figure 2: The HOOMT decomposition process

2.3. The HOOMT Models

The HOOMT methodology currently uses three models, the High Order Object Model (HOOM), the Hierarchical Object Information Flow Model (HOIFM), and the Hierarchical State Transition Model (HSTM). The HOOM will be directly derived by the decomposition of the object diagrams. The HOIFM will be developed based upon the methods of the objects in the HOOM. The HSTM is created to show the dynamic behavior of the objects identified during decomposition. The same levels of abstraction must be present in the three models for consistency checking. Each of the three models presents a different view of the system: the object view, the functional view, and the behavioral view. The primitive objects in the HOOM can then be partitioned into strictly hardware and strictly software components. The HOIFM and the HSTM combine to help define the interfaces between components.

3. Structured Development of the HOOM for Hardware/Software Co-analysis/Co-design

The generation of the HOOM begins with the production of a top level, "black box" view of the overall system object and relevant external objects. For the power system model, this top level, or context object diagram, includes the *FACTS Power System* object, the *Contingency* object, and the *Service Provider (Utility)*

object, and the relationships between these objects (Figure 3). The boxes on the side of the objects show (from top to bottom) the objects' attributes, methods, and constraints. (The addition of the constraints is one of the significant modifications to the model.) It should be noted that the constraints on the operation of the system are here represented as high level requirements such as "voltage stability". The context object diagram captures the interaction of the service provider with the power system, as well as the interaction of errors (or contingencies) on the power system.

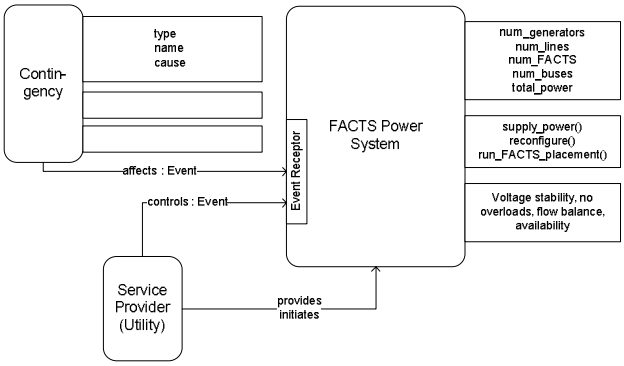

Figure 3: Context object diagram of FACTS power system

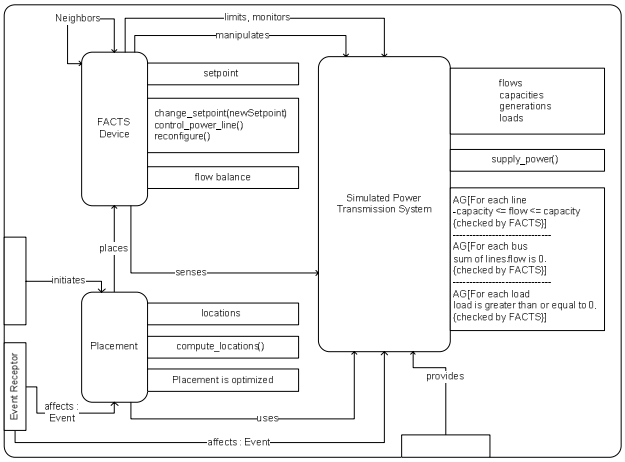

Figure 4: Decomposition of FACTS power system object model

Decomposing the *FACTS Power System* object, we find that it consists of the high order *FACTS Device* and *Simulated Power Transmission System* objects, as well as the primitive *Placement* object (Figure 4). The high level constraints have been decomposed into lower-level constraints. The Placement object represents a design-time activity, initiated by the Service Provider, which determines the optimal placements of FACTS devices into an existing power grid. Around the sides of the FACTS Power System object can be seen various rectangular boxes, or ports. These ports represent interfaces for relationships between external objects and internal objects. These interfaces may be physical, code-based, or simply model-based object interfaces (as a method of retaining the proper abstraction).

The power transmission system may be decomposed into two very different models. One is the actual physical (or deployed) model, consisting of such objects as power lines, power generations, power loads, buses, and sensors, while the other is a hardware/software simulated model. Despite the internal differences, however, the interfaces of the two systems are identical. The target system for this model is the hardware-in-the-loop simulation mentioned above, in which several physical FACTS devices are to be connected to a simulation engine which will calculate the state of the simulated power grid and then simulate that state for the FACTS devices. Therefore it becomes necessary to refine the original *Power Transmission System* object of the model into a *Simulated Power Transmission System* object, maintaining the external interactions as well as the attributes, methods, and constraints of the original object. Figure 5 shows the decomposition of *Simulated Power Transmission System* into the *Simulation Engine* and *HIL Line* objects, and the "set" relationship between the two objects, which represents the power generation settings sent by the Simulation Engine to the HIL Line.

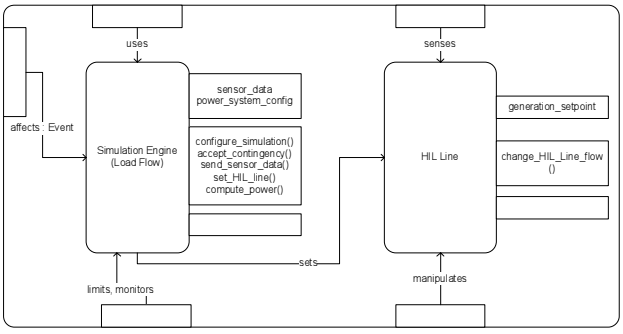

Figure 5: Simulated power transmission system object model

Of particular interest to our research is the *FACTS Device* object. In the model, we have refined the generic *FACTS Device* object into a *UPFC* (Unified Power Flow Controller) *FACTS Device* object due to its use in the target hardware-in-the-loop test system. Figure 6 shows the decomposition of the UPFC FACTS device, in which can be seen the *Embedded Computer*, the *DSP Board*, the *Interface Board*, and the *UPFC Power Electronics*. It is to be noted that the *DSP Board* is represented as replacing the original abstract concept of the "*Interface*" object between the embedded computer and the power electronics. The "limits" and "monitors" relationships seen in Figure 4 continue into the FACTS device, where

they go into the embedded computer. Similarly, the "senses" and "manipulates" relationships are linked to the *UPFC Power Electronics* object. Another salient feature of this object model is the "CAN Bus" connection between the embedded computer and the DSP board. Along this connection data from the DSP board will be passed up to the control programs on the embedded computer, while the control programs will send back the power line control settings to the DSP board, which will then set the power electronics accordingly.

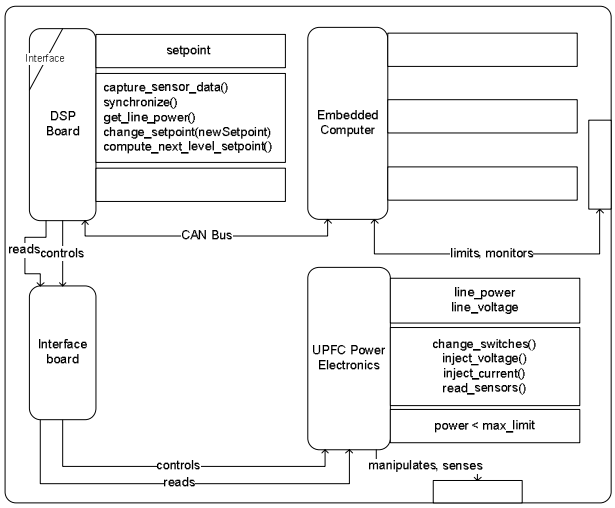

Figure 6: UPFC FACTS device object model

The *Embedded Computer* object contains two of the key software components of the FACTS device, the *Long Term Control* and the *Dynamic Control*. The long term control is a distributed maximum flow algorithm that runs over the FACTS devices placed in a power system [10]. The function of the dynamic control is to respond in the short term to fluctuations in the power line attached to the FACTS device, in addition to providing a smooth transition from the current power settings to new setpoints generated by the long term control.

One of the principle goals of the co-analysis/co-design process using the HOOMT is the structured decomposition of the model to the point where individual components (primitive objects) can be identified as either hardware or software and partitioned accordingly for implementation. If needed, the individual hardware and software components could be further decomposed as necessary, either by continued use of the HOOMT during the co-analysis/co-design phase, or by other means during the implementation phase. For example, once the hardware components have been identified, it may be desirable for the hardware engineers to further represent these components using a language such as VHDL. Primitive objects that represent well-known or pre-existing components may need no further design.

4. Co-analysis and Co-design of the Functionality of the FACTS Power System Hardware and Software using the HOIFM

The HOIFM is used in the HOOMT methodology to represent the functional behavior and data flows of the system being modeled. As with the object model, the methods modeled in HOIFM can be decomposed from the higher levels of abstraction. It is therefore desirable to maintain the same level of abstraction between methods that interact.

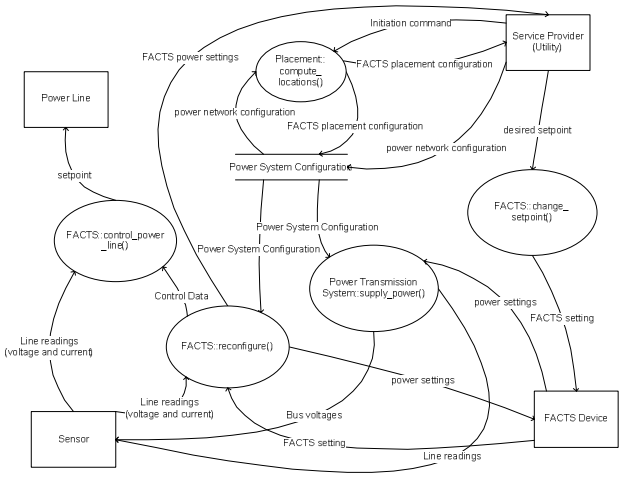

Figure 7: FACTS power system information flows

Decomposition of the methods of the *FACTS Power System* (from Figure 4) reveals the abstraction layer shown in Figure 7, and the lower-level methods that are present: those associated with the FACTS Device, the FACTS placement algorithm, and the (Simulated) Power Transmission System. Further decomposition of the methods is done to analyze their functionalities.

5. Specification of the Behavior of the Hardware/Software of the FACTS Power System with the HSTM

In order to capture the dynamic behavior of the system being modeled, the HOOMT methodology provides the HSTM. As with the HOIFM diagrams, it is important to keep the same levels of abstraction for the state transition diagrams as are found in the HOOM. Figure 8 shows the top level state transition diagram for the FACTS Power System. At this level of abstraction, the states are relatively straightforward. The diagram begins with a power grid without FACTS devices, and transitions to a state in which FACTS placement locations are computed (and FACTS devices are then placed into the power grid). The three main operating states of the system at the top

level are the *Working State* (when the system is stable), the *Reconfiguring* state (initiated typically by a contingency), and a *Degraded Operation* state for when a FACTS device cannot reconfigure the power settings. It should be noted that the Reconfiguring state is a high-level state, requiring further decomposition.

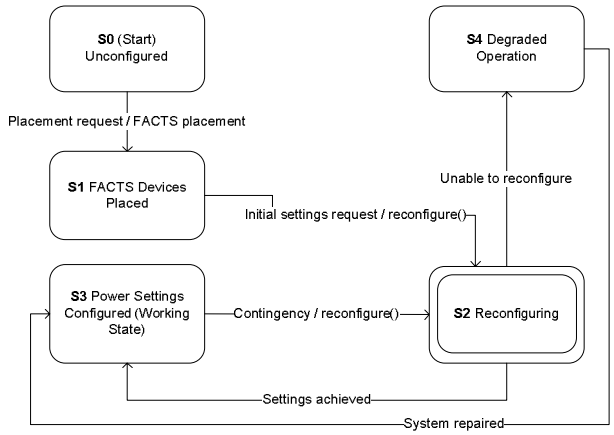

Figure 8: State chart: FACTS power system

Performing further structured decomposition on the Reconfiguring state of Figure 8 leads to some of the more complex interactions of the model.

6. Development Results

The HOOMT models presented in this paper were part of the specifications of the system and its hardware/ software components developed using the HOOMT process discussed in Section 2. Although power grids have been in existence for many years, modeling them, and in particular a FACTS-embedded power system, has never been done using an object-oriented methodology. The models we developed have undergone numerous refinements of the objects, their relationships, and their functionalities as understanding of the advanced power grid control system and its components has increased, and they continue to be refined and further decomposed as necessary.

One particular refinement to the models was the change mentioned in Section 3 from modeling a deployed power transmission system to modeling a simulated power transmission system. This refinement came after much of the initial decomposition and partitioning of many of the system components. The necessary changes to the model affected a number of levels of abstraction.

7. Conclusions

Embedded systems design is becoming increasingly important. However, conventional design approaches have distinct problems, such as a lack of concurrent analysis, lack of a common analysis and design method for hardware and software designers, and the need to determine when and how to partition the system being designed into hardware and software components. The High Order Object-oriented Modeling Technique provides a structured object-oriented methodology for integrated co-analysis and co-design of hardware/ software for an embedded system. It provides a uniform method for specification that allows hardware and software designers to easily collaborate to create the system specifications in a concurrent process. The structured decomposition of the models provides the partitioning of the system by identifying the hardware and software components and their interfaces.

8. 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] R. J. Machado, J. M. Fernandes, and H. D. Santos. "An Object-Oriented Approach to the Co-Design of Industrial Control-Based Information Systems." *4th APCA Portuguese Conference on Automatic Control (CONTROLO 2000)*, pp. 570-575, Guimaraes, Portugal, Oct. 2000.

[3] 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.

[4] 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.

[5] B. P. Douglas. "Doing Hard Time: Developing Real-Time Systems with UML," *Objects, Frameworks and Patterns*. Addison-Wesley, 1999.

[6] X. F. Liu, H. Lin, and L. Dong. "A Systematic Approach to the Integration of Object Modeling with Structured Analysis Based on High Order Object Model." *International Journal of Computer and Information Science (IJCIS)*, 2(2):74-96, June 2001.

[7] G. Hellestrand. "The Engineering of Supersystems." *IEEE Computer*, 38(1):103-105, January 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.