Feature-based Component Model for Design of Embedded Systems

Xuan F Zha, Ram D Sriram

Manufacturing Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899, U.S.A
Email: {Zha, Sriram}@cme.nist.gov

ABSTRACT

An embedded system is a hybrid of hardware and software, which combines software's flexibility and hardware real-time performance. Embedded systems can be considered as assemblies of hardware and software components. An Open Embedded System Model (OESM) is currently being developed at NIST to provide a standard representation and exchange protocol for embedded systems and system-level design, simulation, and testing information. This paper proposes an approach to representing an embedded system feature-based model in OESM, i.e., Open Embedded System Feature Model (OESFM), addressing models of embedded system artifacts, embedded system components, embedded system features, and embedded system configuration/assembly. The approach provides an object-oriented UML (Unified Modeling Language) representation for the embedded system feature model and defines an extension to the NIST Core Product Model. The model provides a feature-based component framework allowing the designer to develop a virtual embedded system prototype through assembling virtual components. The framework not only provides a formal precise model of the embedded system prototype but also offers the possibility of designing variation of prototypes whose members are derived by changing certain virtual components with different features. A case study example is discussed to illustrate the embedded system model.

Keywords: Embedded system, feature-based modeling, component-based approach, UML, object-oriented representation

1. INTRODUCTION

The automotive, telecommunications, multimedia, consumer electronics, industrial automation and health-care industries are witnessing a rapid evolution towards an embedded/coherent solution by implementing complex integration of hardware and software or integrating complete systems on a single chip (SoC). An embedded system is an electronic system embedded within an external process which uses one or more processing elements to assist in performing a function intrinsic to a given product. Modern embedded systems have certain characteristics (ever-increasing complexity and diversity for more functionality, packed into smaller spaces and consuming less power) that demand new approaches to their specification, design and implementation. There exist a large number of informal or semi-formal models and methodologies for separate hardware/software design. Static and traditional partitions of hardware and software and hardware-based notations and approaches are currently used for specification, design and implementation. No unified formal representation, simulation and synthesis knowledge framework is available (Eggermont 2002).

This research focuses on the modeling and representation for design of embedded systems, an initiative under the Manufacturing Interoperability program at NIST. This research aims to develop methodologies, technologies (standards) and framework for modeling and representing information & knowledge in embedded systems design. The scope of research includes hardware/software co-design methodology, integrated framework for design, modeling & simulation, testing, standard representation and protocols for exchanging and reusing system-level information and knowledge to enable semantic interoperability between design software systems in virtual, distributed and collaborative environment through the entire life-cycle.

This paper presents preliminary research results on the design representation of embedded systems. It defines a component-based topology and a feature-based model structure for the integrated representation of components that constitute an embedded system. The feature-based model framework is intended to contain all the information that is required to instantiate in a format suitable to serve as the basis for the representation. Thus, this work contributes to the areas of framework modeling and feature modeling.

The organization of this paper is as follows. Section 2 provides an overall embedded system modeling approach. Section 3 discusses a component-based approach. Section 4 presents the feature-based component

modeling for embedded systems. Section 5 provides an open embedded system feature model (OESFM) based on the UML representation. Section 6 provides a case study. Section 7 summarizes the paper.

2. EMBEDDED SYSTEM MODELING APPROACH

The modeling approach adopted in this research identifies four abstraction levels for the life-cycle development of an embedded system, viz., enterprise, system, component and feature. The enterprise level aims at providing a unified view of the system and its environment by capturing enterprise-related concepts. The system level determines the system being developed, distinguishing it from its environment. The environment of a system consists of information systems or human users that make use of the services provided by the system itself, as well as other systems that provide some service used by the system being developed (de Farias 2001). The component level represents the system in terms of a set of composed components. A component may be further decomposed into sub-components. A composite component is an aggregate of sub-components that, from an external point of view, is similar to a single component. If a composite component is part of a component composition, the design process of this component corresponds to the design process of an isolated system, and the environment of this system contains the other components in the composition. The feature level defines the internal structure of simple components. A component is structured using a set of related features, which are implemented in a feature description and programming language. Thus, the development process of a component at the feature level corresponds to the feature-oriented development process similar to the traditional object-oriented process. The focus of this work is on the component-level and feature-level. More details on the component-level and feature-level modeling will be discussed in the following sections.

3. COMPONENT-BASED SYSTEMS AND COMPONENT MODELING

3.1 Hardware and Software Components

A component is a non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces. In the real world, we easily sense and touch some real hardware component systems such as Lego, mechanical parts, square stones, building plans, electronic components, IC chips and hardware bus. These components generally connect through ports. Figure 1 shows some port-based mechanical and electronic (hardware) components.

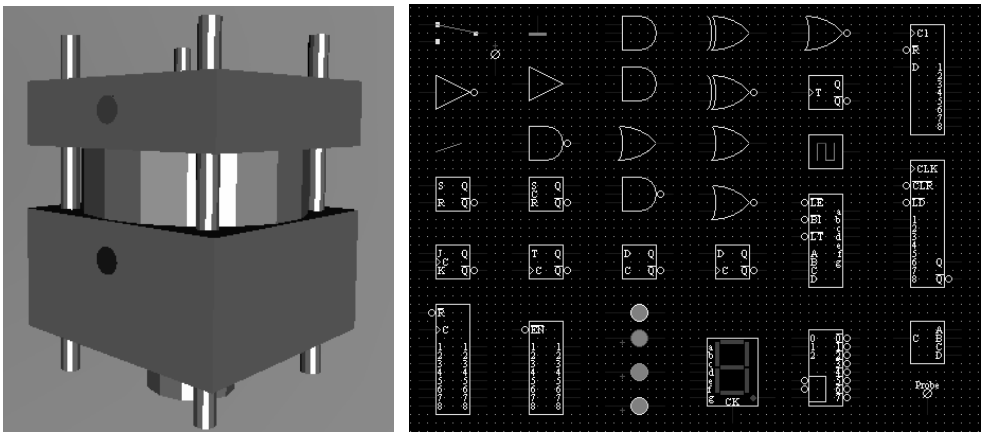

Figure 1: Port-based mechanical and electronic components

A software component is generally a unit of composition with contractually specified interfaces and explicit context dependencies only. It can be deployed independently and is subject to composition by a third party. A run-time software component is a dynamically bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at run-time. The widely accepted software component definition is that a software component is a part of software in a binary form; it is not compiled or rebuilt with

contractually specified interfaces, i.e., defined API and all assumptions in which the component can work. A component can be deployed independently, plugged and played, i.e., it can be dynamically loaded into the system or dynamically replaced. A software component must have a mechanism that makes it possible to compose/integrate the system without need of modifying and rebuilding it. Figure 2 provides two simplified software components.

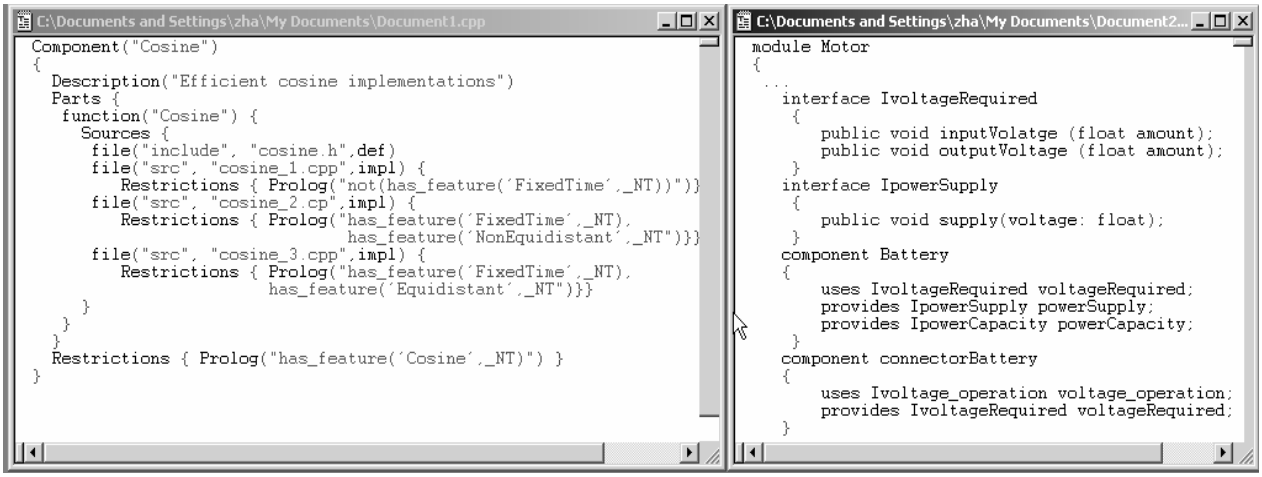

Figure 2: Two simplified software component descriptions

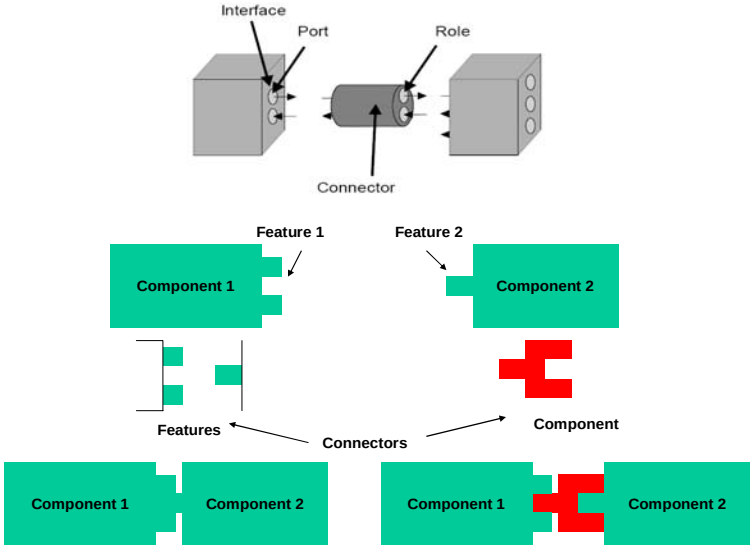

Figure 3: Component-connector model

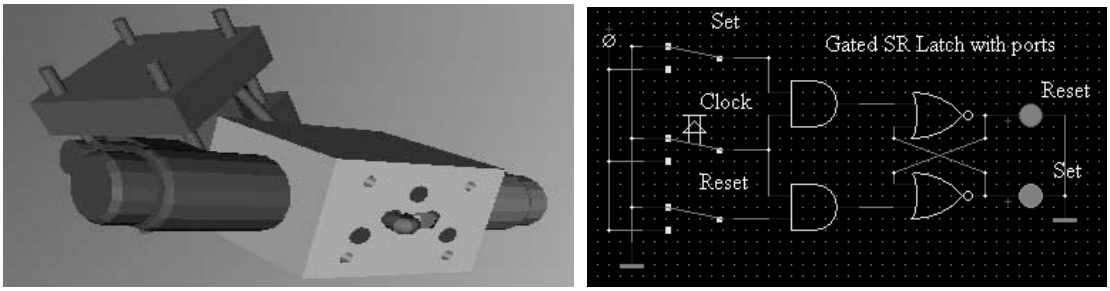

Figure 4: (a) Port-based mechanical assembly and (b) Port-based electronic system schematic

3.2 Component Modeling

The basic concepts in the component-based design approach are components, connectors and systems, in which system is a configuration/assembly. Both components and connectors have connection points called ports for components and roles for connectors (Figure 3). Thus, design elements include component, connector, port, and role. Components are connected to connectors by defining an attachment between the port of a component and the role of a connector. Connectors can be viewed as special communication components. One connector may connect multiple components. Components may be nested but cannot be connected directly to each other and neither can a connector to another connector. Components and connectors have attributes or properties. Properties are un-interpreted values, i.e., they do not have any semantics defined. Figure 4 shows a port-based mechanical assembly for a hydraulic system and a port-based electronic system schematic. In UML 2.0, some new concepts and major improvements have been added to support component-based design. UML 2.0 includes a set of constructs about components and their assembly. Its description may include a set of ports, a set of parts, a set of connectors and a behavior.

4. FEATURE-BASED COMPONENT MODEL FOR EMBEDDED SYSTEMS

Traditionally, the feature model is an input to the design and development phase for software and software families. In this research, instead, the feature model is used to provide a formal description of embedded systems and to formalize knowledge about its instantiation process. Details of the feature-based representation and modeling are discussed below.

4.1 Features and Feature-Based Component Modeling Approach

4.1.1 Feature Models

A feature model can be used to describe the commonalities and differences between the individual hardware/software systems. There are four categories of features (Riebisch 2003, Riebisch et al. 2004):

1) Functional/behavioral features express the behavior or the way users may interact with a system. They describe both static and dynamic aspects of functionality, and may cover use cases, scenarios and structure.
2) Structural features including form features and interface feature expresses the overall form/structure of an embedded system or its HW/SW components and their relationships. Interface features express the system's conformance to a standard or a subsystem. They describe connectivity and conformance aspects as well as contained components.
3) Parameter features express enumerable, environmental or nonfunctional properties. They cover all features with properties demanding quantification by values or for assignment to quality, e.g., color.
4) Concept features represent an additional category for structuring a feature model. They encapsulate abstract features within a hierarchical features structure. The root of the hierarchy always represents a concept feature. Features in this category have no concrete implementation, but their sub-features provide one.

Within a feature model, the features are structured by relationships. Common to all methods mentioned above are hierarchical relationships between a feature and its sub-features. The hierarchical relationships control the inclusion of features to instances. If an optional feature is selected for an instance, then all mandatory sub-features have to be included as well, and optional sub-features can be included.

4.1.2 Feature Modeling with UML

Feature modeling is the activity of modeling features and their interdependencies and organizing them into a feature model. It provides a model of end-user visible features that are present in a given domain by providing a description for each feature and a relationship model for these features. Feature modeling approaches are usually based on a two-layer structure: 1) a meta-modeling level, which defines the types of features that can be used, their properties, and mutual relationships, and 2) a modeling level where the feature model for the entities of interest is constructed. Feature models require the definition of a concrete syntax and language to express them. The application feature model is seen as an instance of a feature meta-model (Grades 2003). In this research, we propose UML-based formalisms to represent the feature meta-model. Figure 5 shows a UML diagram of the feature meta-model. The basic ideas can be summarized as follows. A feature can have sub-features, but the connection between a feature and

its sub-features is mediated by the group. A group gathers together a set of features that are children features of some other features. Thus, a group represents a cluster of features that are children of the same feature and that obey some constraints on their legal combination. Groups are also used to enforce local restrictions (constraints). The same feature can belong to several groups. Both features and groups have cardinalities. The cardinality of a feature defines the number of instances of the feature that can appear in an application. The cardinality of a group defines the number of features chosen from within the group that can be instantiated in an application. Cardinalities can be expressed either as fixed values or as ranges of values. The application feature model is instantiated from the meta-model.

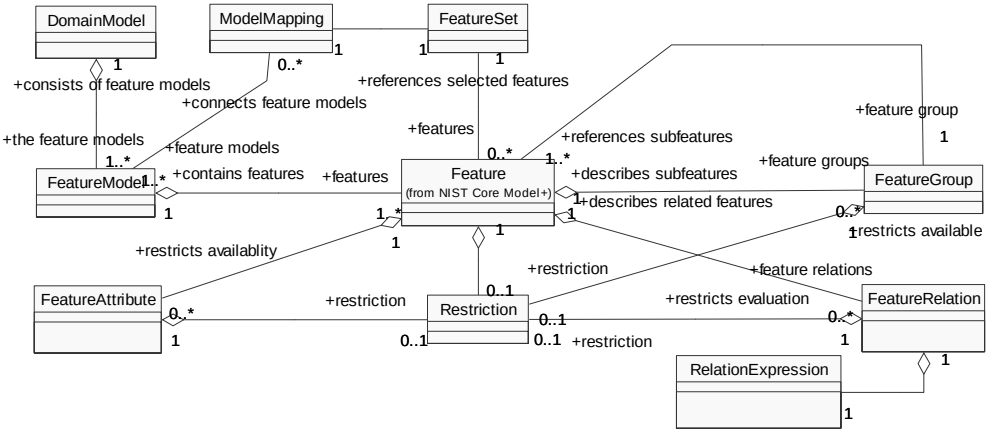

Figure 5: UML class diagram for feature meta model

4.1.3 Feature-Component Correspondence

The feature-oriented reuse method (FORM) (Kang et al. 1998) is an extension of feature-oriented design and analysis (FODA) that includes architecture design and object-oriented component development and assists in developing reusable and adaptable artifacts from product features (Kang et al. 2002). FORM begins with feature modeling, where the resulting feature model serves as a basis for reusable and adaptable artifacts. During the architecture design activity, features are allocated to architectural components and the dependencies between them are specified. The functional architecture, constituting the architectural components, is refined into process and deployment architectures, which are used during component design. In implementation, the technique described here is based on the design and execution traces generated by a profile for different usage scenarios (Eisenbarth 2001). One scenario represents the invocation of a single feature or a set of features and yields all artifacts/sub-artifacts (e.g., sub-programs as sub-software artifacts) executed for these features. These artifacts/sub-artifacts (sub-programs) identify the components (or are themselves considered components) required for certain features. The required components for all scenarios and the set of features are then subject to concept analysis. Concept analysis gives information on relationships between features and required components.

4.2 Feature-Based Component Model for Embedded Systems

4.2.1 Hardware and Software Components in Embedded Systems

Typically, an embedded system is housed on a single microprocessor board with the software (programs) stored in some form of read-only memory, such as ROM, EPROM, or flash memory. Embedded systems do not use conventional I/O devices such as a keyboard, a mouse, and a display. Instead, they interact with the outside world (environment) through their sensors and actuators. Sensors feed the input data to the system and actuators deliver the output to the external environment. Embedded system software can generally be classified into the following three categories according to the problem solving methods used (Hassani 2000): 1) numerical or data processing, 2) user interface, and 3) decision making. The numerical or data processing software is used in problems that have numerical solutions. The output response can be calculated as a mathematical function of inputs. The problem solving software is made up of a few modules that use numeric equations to produce the results. The user interface

module is used for facilitating data/message passing for users. Decision-making schemes are generally applied to problems that do not have numerical solutions. Instead, they use large number of If-THEN statements, monotonic logic, and heuristics to achieve reasonable solutions. The decision-making module typically consists of rules. Rules are sets of conditional statements with IF-THEN structure that logically relates information contained in the condition element (IF part) to other information contained in the action element (THEN part).

4.2.2 Embedded System Component Features

Hardware and software features compose hardware and software components, respectively. This means that feature configurations determine the underlying SW and HW components. As discussed above, features are classified into four categories: concept feature, function (behavioral) feature, parameter feature and structural feature (interface feature, or port). Thus, these four category features compose both a hardware feature and software feature correspondingly, which means that the hardware feature may be specialized into HW concept feature, HW function (behavioral) feature, HW parameter feature and HW interface feature; the software feature generalizes SW concept feature, SW function (behavioral) feature, SW parameter feature and SW interface feature. Normally, interface features are also called port, thus, we may have a SW port and a HW port, accordingly in the software and hardware features. SW port is specialized into Input Port (Requested Port), Output Port (Provided Port), In-Out Port, (Resource Port, and Configuration Constants) (Stewart et al 1993); HW Port is specialized into Input Port (Destination Port, Requested Port), Output Port (Source Port, Provided Port), etc.

Embedded system connectors represent the connections between HW/SW components or subsystems in the embedded system. Connectors may be either HW/SW features or HW/SW components or HW/SW subsystems composed of HW/SW features (or HW/SW components). The embedded system connector can be specialized into subclasses: hardware connector, software connector and hardware-software connector. The hardware connector represents the connections between hardware components or subsystems in the embedded system. The software connector represents the connections between software components or subsystems in the embedded system. The hardware-software connector represents the connections between hardware and software components or subsystems in the embedded system. Differing from those interface features of HW/SW components, interface features of HW/SW/HW-SW connectors are sometimes called roles.

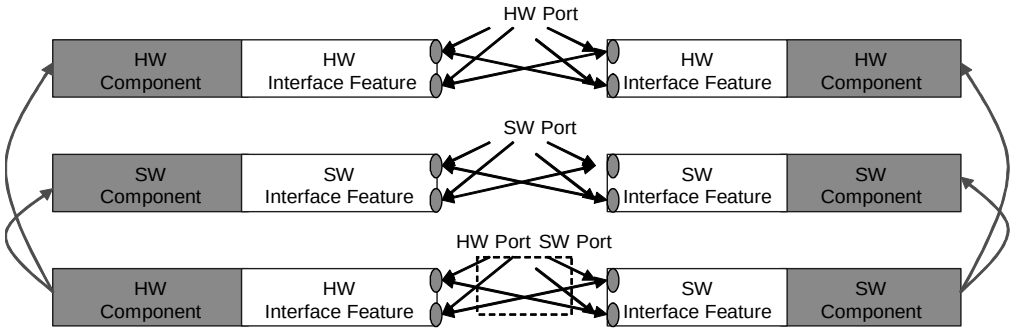

Figure 6: Feature interaction scenario in the embedded system

The scenario of feature interactions in an embedded system can be described as in Figure 6. We propose an approach for modeling feature (port) interactions to comply with the component-connector model based on UML 2.0. We also model feature interactions with feature-solution graphs (Bruin and Vliet 2001). The key idea is to connect features (i.e., user requirements) with solution fragments in a so-called feature-solution graph. This graph serves two purposes. First, it can be used to pinpoint feature interactions. Second, it can guide an iterative architecture development and evaluation process. Feature space consisting of feature models describes the desired properties of the system as expressed by the user. Solution space contains the internal system decomposition in the form of a reference architecture composed of components. In addition, the solution space may also contain general applicable solutions that can be selected to meet certain non-functional requirements.

4.3 Feature-based Component Model Representation with XML

To advance the development of computer-aided design (virtual prototyping) tools for embedded systems, there is a need to address the formal representations of different abstractions of behavioral, structural, and product data along with their integration and exchange. Based on the formalism of unified feature component model and the taxonomy, we separate the information required for each standard/custom component into three distinct groups: functional/behavioral attributes, structural attributes, and parameter attributes. XML schema and XML are used for attribute-based feature representation in this research. The XML schema is transformed from the UML models.

5. UML REPRESENTATION FOR EMBEDDED SYSTEM FEATURE MODEL

An Open Embedded System Model (OESM) is currently being developed at NIST to provide a standard representation and exchange protocol for embedded systems and system-level design, simulation, and testing information (http://www.mel.nist.gov/proj/pe.htm). In this section, we only discuss in detail the embedded system feature model in OESM, i.e., Open Embedded System Feature Model (OESFM), related to models of embedded system artifacts, embedded system components, embedded system features, and embedded system configuration/assembly. We use UML notation and diagrams to explain the embedded system feature model.

5.1 The NIST Core Product Model Extension to Embedded Systems

NIST research efforts toward the development of the basic foundations for the next generation CAD systems lead to the NIST Core Product Model (CPM) (Fenves 2001). However, CPM currently focuses mainly on the physical artifact (e.g., motor, airplane), especially for electro-mechanical products or assemblies. There is a need to make some modifications/extensions for it to be used for an informational artifact (e.g., software, organizations, business processes, plans and schedules). Consequently, CPM needs some modifications/extensions when applied for modeling embedded systems. The modification/extension of CPM package includes expanding semantically the definitions of some concepts and/or extending existing classes or adding new classes. For more information on the CPM, please refer to (Fenves 2001).

According to the extension of NIST-CPM, **ES_Artifact** refers to an embedded system or one of its hardware/software (HW/SW) components. **ES_Artifact** is specialized into two classes: **HardwareArtifact** and **SoftwareArtifact**. **HardwareArtifact** refers to a hardware system/component in an embedded system, which is an aggregation of **HardwareFunction**, **HardwareForm** and **HardwareBehavior**. **HardwareFunction** represents what the artifact is supposed to do; **HardwareForm** represents the proposed design solution for the design problem specified by the hardware function; and **HardwareBehavior** represents how the hardware artifact realizes its function. **HardwareForm** itself is the aggregation of **Geometry**, the spatial description of the artifact, and **Material**, the internal composition of the hardware artifact. **HardwareFeature** represents any information in the **HardwareArtifact** that is an aggregation of **HardwareFunction** and **HardwareForm**. **SoftwareArtifact** refers to a software system in the embedded system or one of its software components, i.e., which is an aggregation of **SoftwareFunction**, **SoftwareForm** and **SoftwareBehavior**. **SoftwareFunction** represents what the software artifact is supposed to do; **SoftwareForm** represents the proposed solution for the design problem specified by the software function; **SoftwareBehavior** represents how the software artifact implements its function. **SoftwareForm** itself is the aggregation of **Architecture**, the structural description of the software artifact, and **Code**, the internal composition of the software artifact. The class **Code** is also specialized into two subclasses: **SourceCode** and **BinaryCode**. **SoftwareFeature** represents any information in the **SoftwareArtifact** that is an aggregation of **SoftwareFunction** and **SoftwareForm**. All the above entities have their own independent containment ("part-of") hierarchies.

5.2 Embedded System Feature Model

Figure 7 shows the main schema of the Open Embedded System Feature Model (OESFM). The schema incorporates information about design specification, partitioning, embedded system specification, and component composition and configuration/assembly relationships (Zha and Du 2002, Zha et al. 2004). The model incorporates information about component composition (part-of) and assembly/configuration relationship. The component composition of an embedded system is modeled using this part-of relationship. An embedded system represented by the **EmbeddedSystem** class is decomposed into hardware/software (HW/SW) subsystems and components, and

connectors connecting theses subsystems and components. Each embedded system component represented as **ES_Component** class in the **ES_Component** package, whether a HW/SW sub-system or component, is made up of one or more HW/SW features, represented in the model by **ES_Feature** class in the **ES_Feature** package. The **EmbeddedSystem** and **ES_Component** classes are subclasses of the **ES_Artifact** class (extended from NIST-CPM **Artifact** class, see above). **ES_Feature** is a subclass extended from the NIST-CPM **Feature** class. The composition (configuration/assembly) relationship is represented by a class named **CompositionAssociation**. Components or subsystems in the embedded system are connected by connectors represented by **ES_Connector** class in the **ES_Connector** package. Connectors may be either features or components or subsystems composed by features or components. We only summarize some of them below.

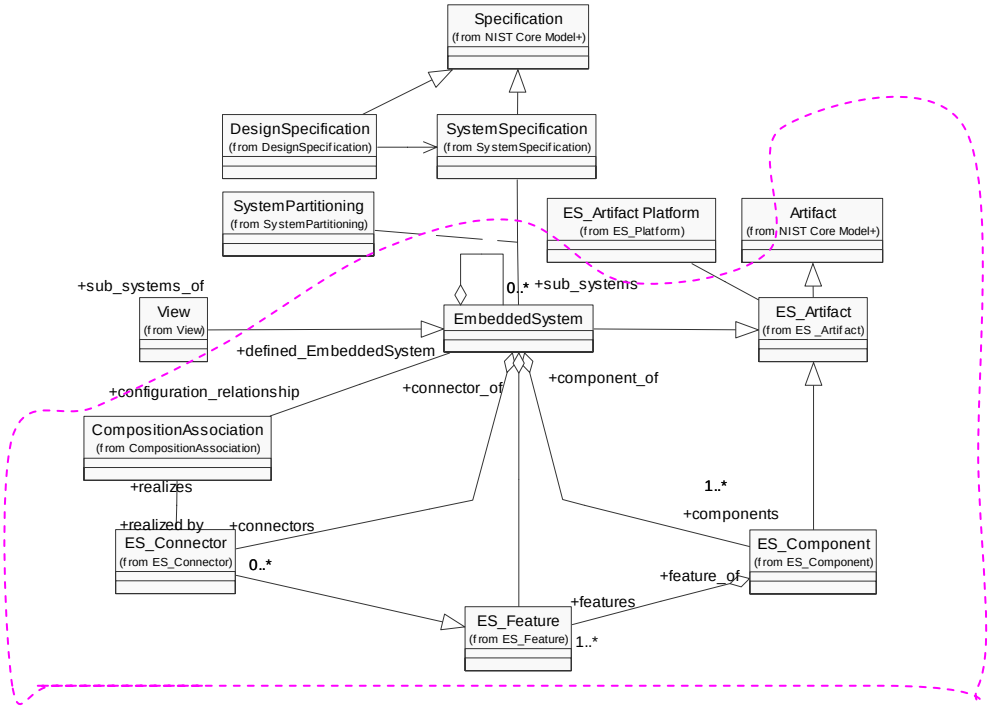

Figure 7: Main schema of the Open Embedded System Model

The class **ES_Component** represents embedded system component, which is a composition of **ES_FunctionFeature, ES_ConceptFeature, ES_ParameterFeature** and **ES_StructuralFeature**. It is specialized into **HardwareComponent** and **SoftwareComponent.** Thus, **HardwareComponent** is an aggregation of **HardwareFunctionFeature, HardwareConceptFeature, HardwareParameterFeature** and **HardwareStructural Feature**; **SoftwareComponent** is an aggregation of **SoftwareFunctionFeature, SoftwareConceptFeature, SoftwareParameterFeature** and **SoftwareStructuralFeature**. **HardwareInterfaceFeature** is a specialization of **HardwareStructuralFeature**; **SoftwareInterfaceFeature** is a specialization of **SoftwareStructuralFeature**.

The class **ES_Feature** (Figure 8) is a sub-class of the **Feature** class defined in NIST-CPM. It inherits the function and form information from **Feature**. **ES_Feature** is specialized into the following subclasses: **ES_FunctionFeature, ES_ConceptFeature, ES_ParameterFeature,** and **ES_InterfaceFeature**. **ES_Feature** has three subclasses: **HardwareFeature**, **SoftwareFeature**, and **CompositeFeature**. **CompositeFeature** represents a composite feature that can be decomposed into multiple simple features. **SoftwareFeature** is specialized into **SoftwareFunctionFeature, SoftwareCeonceptFeature, SoftwareParameterFeature,** and **SoftwareInterfaceFeature**. **SoftwareInterfaceFeature** is specialized into **SoftwarePort. SoftwarePort** is specialized into **InputPort**, **OutputPort**, and **InOutPort. HardwareFeature** is specialized into **HardwareFunctionFeature, HardwareCeonceptFeature, HardwareParameterFeature,** and **HardwareInterfaceFeature**. **HardwareInterfaceFeature** is specialized into **HardwarePort. HardwarePort** is specialized into **InputPort** and **OutputPort.** The class **ES_InterfaceFeature** is specialized into two subclasses:

HardwareInterfaceFeature and **SoftwareInterfaceFeature.** The class **ES_InterfaceFeatureAssociation** refers to the composition/assembly relationship between one or more embedded system interface features. This relationship is represented by the class **ES_InterfaceFeatureAssociationRepresentation**. The diagram also shows that the **ES_Artifact** Association is the aggregation of **ES_InterfaceFeatureAssociation.**

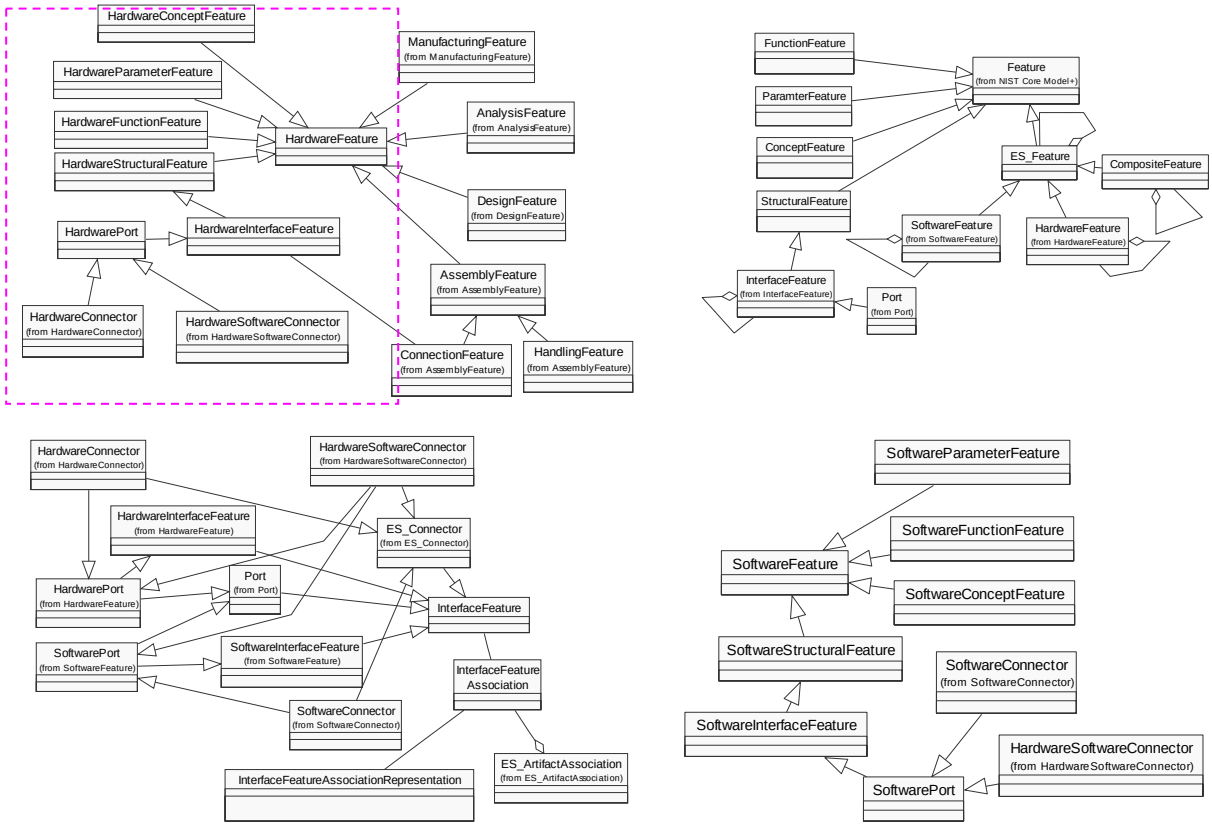

Figure 8: Class diagram of embedded system features (HW features, SW features, interface features)

The class **ES_Connector** represents the connections between components or subsystems in the embedded system. Connectors may be either features or components or subsystems composed of features or components. **ES_Connector** is specialized into subclasses**: HardwareConnector, SoftwareConnector** and **HardwareSoftwareConnector. The HardwareConnector** represents the connections between hardware components or subsystems in the embedded system. **SoftwareConnector** represents the connections between software components or subsystems in the embedded system. **HardwareSoftwareConnector** represents the connections between hardware and software components or subsystems in the embedded system.

6. CASE STUDY

In this section, we devise a simple hydraulic measurement and control system (HMCS) as a case study to illustrate the feature-based component model discussed above. This example is inspired by a weather station system described in (Grades 2003). The goal is to develop a complete hydraulic measurement and control station for testing/diagnosing car antilock braking systems (ABS) based on a small experimental microcontroller ATMEL ATMEGA103. The microcontroller board is equipped with several sensors (pressure, temperature, speed) and has an LCD display, a serial controller, a USB controller, and Modem/Internet controller for output and input purposes. Figure 9 shows the schematic of the hydraulic measurement and control system. Table 1 gives a partial list of the components features. The resulting concept feature model for this application is shown in Figure 10, including feature space exploration, feature configuration, feature diagram and its UML representation. Based on these

features, HMCS can be designed by the developed prototype system for feature-based embedded and mechatronic system virtual prototyping. This prototype system incorporates the feature-modeling tool, CaptainFeature.

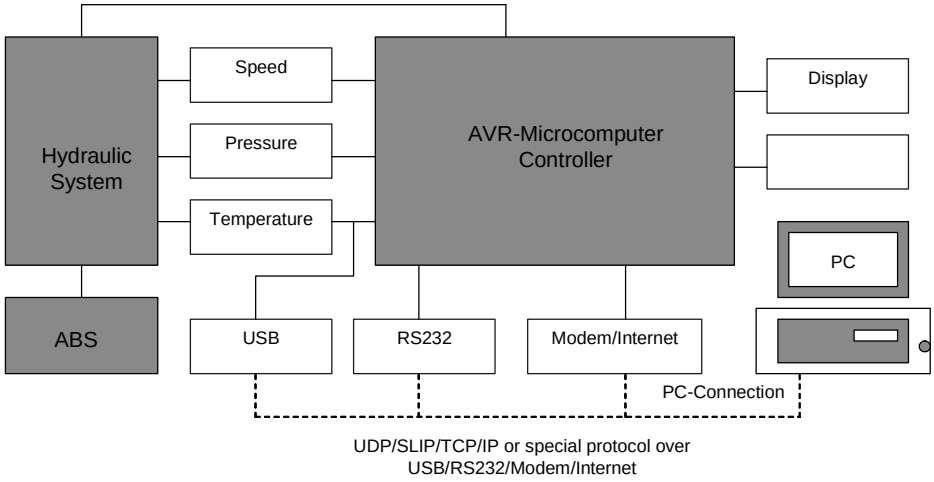

Figure 9: The hydraulic measure and control system

Table 1: Partial component feature list of the system

ID	Component Name	Feature Descriptions			
		Concept Feature	**Functional/Behavioral Feature**	**Structural Feature (Interface)**	**Parameter Feature**
1	Hydraulic system	Actuator, power supply	Speed, pressure, tempature	Input port, output port	Cost $, weight, size
2	Pump	Power supply	Power, pressure, speed, flow. Etc.	Input port, output port	Cost $, weight, size
3	Valve	Flow control	Viscosity, pressure, ambient temperature, max flow, etc.	Input port, output port	Cost $, weight, size
4	Cylinder	Actuator, force transmission	Load (transmission force limit), etc.	Input port, output port	Cost $, weight, size
5	Microcomputer controller	Processor and controller	Memory size, etc	64 Pin TQFP (Input port, output port)	Cost $, size
6	Speed sensor	Input (speed)	Resolutions	Input port, output port	Cost $, size
7	Pressure sensor	Input (pressure)	Resolutions	Input port, output port	Cost $, size
8	Temperature sensor	Input (temperature)	Resolutions	Input port, output port	Cost $, size
9	LCD Display	Output (formatted and unformatted)	Resolutions	Input port	Cost $, size
10	RS232 –Serial	Output(formatted, UDP, unformatted)			
11	RS 232 driver	Output (interrupt operation, change parameters, SCC, SCI, UART)	Size, running speed, etc	Input port, output port	File Size
12	USB protocol	Output (formatted and unformatted)	Transmission rate, etc.		
13	Modem/Internet protocol	Output (formatted, TCP/IP, unformatted)	Connection speed and Transmission rate, etc.		
…	…	…	…	…	…

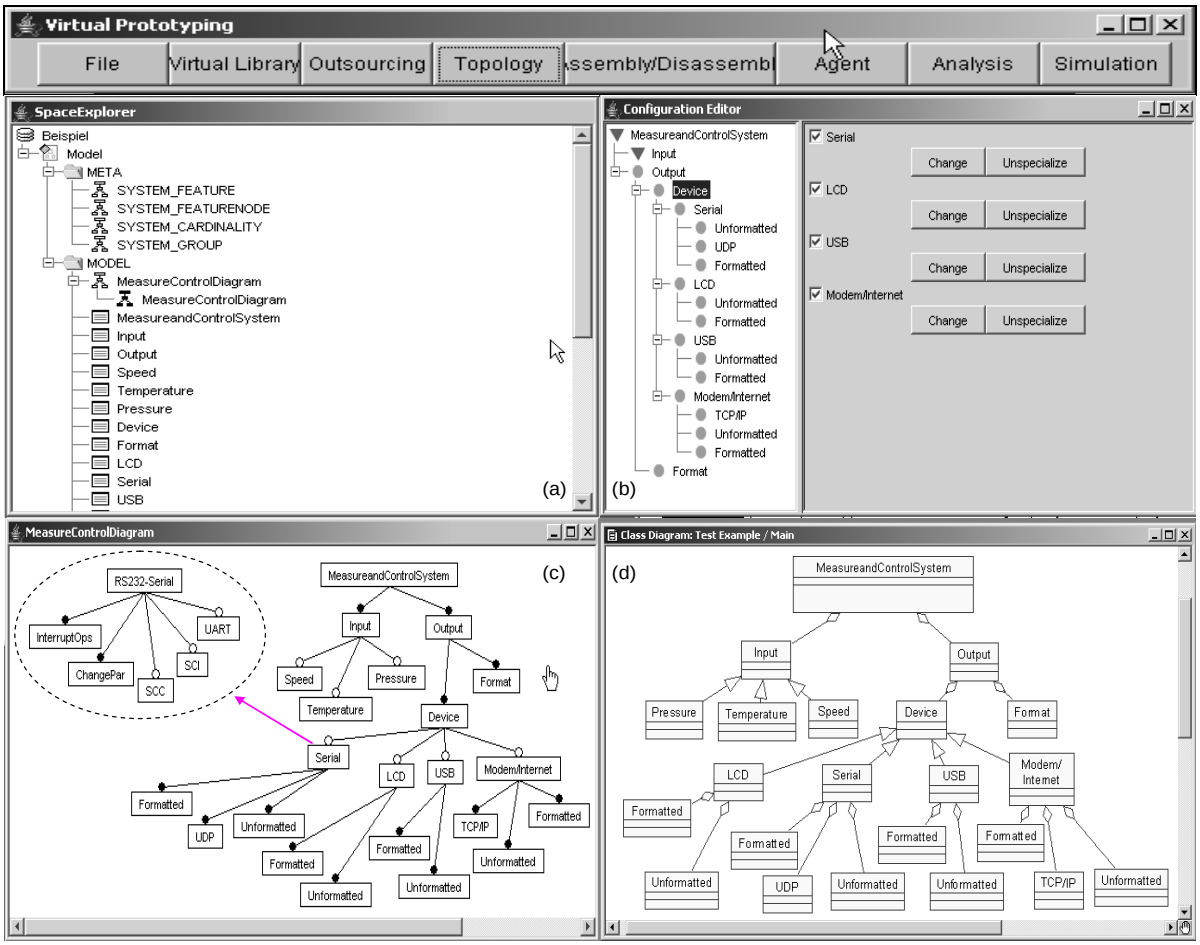

Figure 10: The application feature models: (a) feature space exploration, (b) feature configuration, (c) feature diagram and (d) feature UML diagram

7. CONCLUSIONS AND FUTURE WORK

In this paper, we described a preliminary object-oriented UML representation of a feature model for embedded systems (OESFM). This model incorporates component representation, interface, interaction, assembly relationships, and embedded system features. The OESFM is based on the modification/extension of the NIST Core Product Model (CPM). The classes defined in OESFM, for example embedded system containing hardware and software artifacts, generally inherit function, behavior, and form from the Artifact class of CPM but modified/extended to some extent. The UML model of the embedded system features is described with an example. Currently, we are consolidating this model to be used as one part of the Open Embedded System Model (OESM). Further we will make it harmonized with other models and interoperate with various EDA systems and also explore the possibilities of integrating it with virtual prototyping systems based on feature-based component agent technology.

Acknowledgement
The authors wish to appreciate the valuable comments and improvements suggested by Prof. Steven J. Fenves. Those comments have substantially improved and shaped the paper.

Disclaimer
This “staged” prototype, high-level, logical data model is prepared for analysis and general exchange interest purposes only. This paper should not be interpreted as a proposal to use the prototype presented for the detailed

design of the actual, final logic model for embedded system; implementation models supporting those tasks may or may not incorporate any of concepts presented. No approval or endorsement of any commercial product, service or company by the National Institute of Standards and Technology is intended or implied.

REFERENCES

1. Aßmann, U., Schmidt,R. (1997), Towards a model for composed extensible components, Workshop Foundations of Component-Based Systems, Proceedings, Zurich, Switzerland
2. Beuche, D., Papajewski, Holger, Schroder-Preikschat, W. (2004), Variability management with feature models, Science of Computer Programming, Elsevier Publishers, to appear
3. Beuche, D. (2001), Feature based composition of an embedded operating system family, Proceedings of ECOOP 2001 Workshop #08 Feature Interaction in Composed System, In Association with the 15th European Conference on Object-Oriented Programming, Budapest, Hungary
4. Berg, Kathrin Müller, John, Bishop, Judith, and Zyl, Jay van (2004), The use of feature modeling in component evolution, Technical Report, http://polelo.cs.up.ac.za/publications.htm
5. Bruin, Hans de and Vliet, Hans van (2001), Feature and feature interaction modeling with feature-solution graphs, 2001
6. de Farias, C. R Guareis., Ferreira Pires, L., van Sinderen, M., and Quartel, D., A combined component-based approach for the design of distributed software systems, wwwhome.cs.utwente.nl/~pires/publications/ftdcs2001.pdf
7. Deursen, Arie van and Klint, P. (2001), Domain-Specific Language Design Requires Feature Descriptions, CWI Report, SEN-R0126, ISSN 1386-369X
8. Eggermont, L. D.J. (ed.) (2002), Embedded Systems Roadmap 2002, Vision on Technology for the Future of PROGRESS, 30 March
9. Eisenbarth, T., Koschke,R., and Simon, D. (2001), Feature-driven program understanding using concept analysis of execution traces, Proceedings of the International Workshop on Program Comprehension (IWPC'01), May 12-13, Toronto, Canada
10. Fenves, S. (2001), A Core Product Model for Representing Design Information, NISTIR 6736, NIST, Gaithersburg, MD.
11. Grades, zur Erlangung des akademischen (2003), Composition and Construction of Embedded Software Families, PhD Dissertation der Otto-von-Guericke-Universität Magdeburg, Germany
12. Gurp, Jilles van, and Bosch, Jan, Managing variability in software product lines, Landelijk Architectuur Congres, Amsterdam 2000
13. Hassani, Mehrdad, A Component-based Methodology for Real-time Decision-making Embedded Systems, PhD Dissertation, University of Maryland, 2000
14. Crnkovic, Ivica and Larsson, M. (2001), Component-based software engineering – new paradigm of software development, Invited Talk & Invited Report, Proceedings of MIPRO 2001, Opatija, Croatia
15. Jansen, Anton, Smedinga, Rein, Gurp, Jilles van, and Bosch, Jan, Feature-based product derivation, http://www.cs.rug.nl/~rein/publications/FeatureComposition.pdf.
16. Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M. (1998), FORM: A feature-oriented reuse method with domain-specific reference architectures, Annals of Software Engineering, Vol.5, J. C. Baltzer, AG Science Publishers, Red Bank, NJ, USA, pp. 143-168.
17. Kang, K. C., Lee, J., Lee, K. (2002), Feature Oriented Product Line Software Engineering: Principles and Guidelines, Chapter 2, Domain Oriented Systems Development: Perspectives and Practices, Taylor & Francis, UK
18. Pashov, I., and Riebisch, M. (2004),Using feature modeling for program comprehension and software architecture recovery, Proceedings of 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS2004), Brno, Czech Republic, pp. 406-117
19. Rastofer, U. (2002), Modeling with components– towards a unified component meta model, Proceedings of ECOOP 2002 Workshop #12 Model-based Software Reuse, Malaga, Spain
20. Riebisch, M. (2003), Towards a more precise definition of feature models, Modeling Variability for Object-Oriented Product Lines, M. Riebisch, J. O. Coplien, D, Streitferdt (Eds.), Book On Demand Publ. Co., Norderstedt, pp. 64-76.
21. Riebisch, M., Streitferdt, D., and Pashov, I. (2004), Modeling variability for object-oriented product lines, Buschmann, Frank; Buchmann, Alejandro P.; Cilia, Mariano (Ed.): Object-Oriented Technology. ECOOP 2003 Workshop Reader. Springer, Lecture Notes in Computer Science, Vol. 3013, pp. 165 - 178.
22. Stewart,D.B., Volpe, R.A., and Khosla, P.K. (1993), Integration of real-time software modules for recogfiguration sensor-based control systems, Proceedings of International Symposium on Intelligent Robotics (ISIR'93), Bangalore, India
23. Tierney, P.J. and Ajila, S. A. (2002), FOOM - Feature-based object oriented modeling: implementation of a process to extract and extend software product line architecture, PDSTD'02 – SCI2002 / ISAS2002, July 14 – 18, 2002, Orlando, USA.
24. Zha, X.F., and Du, H. (2002), A PDES/STEP based model and system for concurrent integrated design and assembly planning, Computer-Aided Design, 34 (12), pp. 1087-1110
25. Zha, X.F., Sebti, F., Sudarsan, R., and Sriram, R.D. (2004), Analysis and evaluation of STEP-based electro-mechanical assemblies: an integrated fuzzy AHP approach, Proceedings of ASME DETC 2004, Paper No. DETC CIE-57708