Submitted to Proceedings of DETC'2006
2006 ASME Design Engineering Technical Conference
September 10-13, 2006, Philadelphia, PA, USA

DETC2006-99543

Feature Technology and Ontology for Embedded System Design and Development

Xuan F. Zha
National Institute of Standards and Technology
Gaithersburg, MD and University of Maryland,
College Park, MD 20742

Ram D. Sriram
National Institute of Standards and Technology
Manufacturing System Integration Division
Gaithersburg, MD 20899

Abstract – In this paper, we present our recent effort on using a feature technology and ontology for embedded systems modeling and design. We present an overview of embedded system design and propose an object-oriented UML modeling approach to representing embedded systems, i.e., open embedded system model (OESM). OESM supports models of embedded system artifacts, components, features, configuration/assembly, and embedded system platform and family, design rationale, etc. Our focus is on modeling of feature semantics in embedded systems. We call this open embedded system feature model (OESFM). We also present a semantic web environment for modeling and verifying feature models using ontologies, in which the Protégé-OWL is used to precisely capture the relationships among features in feature diagrams and configurations. The OESFM models and ontologies provide a feature-based component collaborative framework. This allows the designer to develop a virtual embedded system prototype through assembling virtual components in which the platform-based hardware/software (HW/SW) co-design is supported and the design rationale is captured. The collaborative co-design framework can not only provide formal precise models of the embedded system prototypes but also offers design variation of prototypes whose members are derived by changing certain virtual components with different features.

Keywords: Embedded Systems, Feature Model, Semantic Web, Hardware and Software Co-design, Platform-based Design, Embedded System Family, Design Rationale, UML, Ontologies, and Protégé-OWL

1 Introduction

Many industries (e.g., automotive, telecommunications, multimedia, consumer electronics, industrial automation, and health-care) are witnessing a rapid evolution toward solutions that integrate/embed hardware and software or incorporate complete systems on a single chip (SoC). "An embedded system is a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. In some cases, embedded systems are part of a larger system or product, as in the case of an antilock braking system in a car."(http://en.wikipedia.org/). An embedded system is a hybrid of hardware and software, which combines software's flexibility and hardware real-time performance. Modern embedded systems have characteristics (including ever-increasing complexity and diversity for more functionality, packed into smaller spaces consuming less power) that demand new approaches to their specification, design and implementation.

An embedded system information model contains information regarding hardware and software components and their composition/assembly relationships as well as the information related to design and manufacturing. Hence, the modeling approach needs to emphasize the nature and requirements of information for HW/SW components and their assembly relationships. Furthermore, the modeling approach needs to address the evolution of corresponding information models during the conceptual and detailed design stages of embedded systems. Feature technology

(including feature definitions, modeling and representations, constraints, validation, language, and ontology) has been recognized as an emerging means for modeling and design of embedded systems. Formally, a feature is defined as a portion or aspect of the artifact's form that has some specific function assigned to it (Shah and Mantyla 1995, Fenves et al. 2005). A feature has its own containment hierarchy, so that compound features can be created out of other features. Features can be a great help in speeding up design and in guaranteeing manufacturability of individual features on individual components. A feature model that gives a hierarchical structure to the features can be used to describe the commonalities and differences between the individual hardware/software (HW/SW) systems and electro-mechanical systems and integrate them. The features in an embedded system can make it different from any other systems. A new way of looking at a system can be represented by adding a new view of the features of the system. This provides uniqueness and strength of the feature-based embedded system design method that does not exist in other design methods.

This paper aims to present our recent effort in use of feature technology and ontology to embedded systems modeling and design. The organization of this paper is as follows. Section 2 reviews previous work and current research status. Section 3 provides an overview of embedded system design. Section 4 presents an open embedded system model (OESM) and its representation. Section 5 presents the feature-based component modeling for embedded systems and provides a feature semantic model for embedded systems. Section 6 discusses platform-based design by feature. Section 7 summarizes a feature-based approach to HW/SW co-design. Section 8 discusses feature-based embedded system design rationale. Section 9 proposes a feature-based service-oriented distributed collaborative design framework for embedded systems. Section 10 provides a case study. Section 11 concludes the paper.

2 Current Research Status

There exist a large number of informal or semi-formal models and methodologies for separate hardware/software design and several frameworks and tools have been proposed and developed (Eggermont 2002). Static and traditional partitions of hardware and software and hardware-based notations and approaches are currently used for specification, design, and implementation. There is as yet no unified formal standard representation, simulation, and synthesis framework for supporting hardware and software co-design in a distributed collaborative environment despite the progress made in several research projects such as Ptolemy (Lee 2003) and Metropolis (Balarin et al. 2003, Chen et al. 2002). Table 1 summarizes the current research status in embedded and hybrid systems design, including models, methodologies and tools, and standards (Eggermont 2002, Balarin et al. 2003, Chen et al. 2002, Esser 1996) [1]. Other research activities in this area include: 1) some fundamental work on domain-specific heterogeneous & embedded systems modeling and representation; 2) limited industrial vendor activities for real-time systems design and development; and 3) limited university research on hardware/software co-design and platform-based design.

At NIST, we have developed a standards-based formal framework for modeling information and knowledge in embedded systems design so as to enable semantic interoperability between design software systems in virtual, distributed and collaborative environments through the entire lifecycle (Zha, Fenves and Sriram 2005a, 2006). Our work includes HW/SW co-design methodologies, an integrated framework for design, modeling and testing, and standard representations and protocols for exchanging and reusing system-level information and knowledge. Research activities focused on embedded systems modeling and representation and platform-based design. Specifically, the activities include: system partition and modularization, feature-based HW/SW component model, architectural product platform model, decision (evaluation and selection) of components in the configuration and integration of system models, hardware and software co-design, and the distributed collaborative design and (re)configuration framework. We developed an object-oriented UML (Unified Modeling Language) representation of feature models for embedded systems (OESFM). Our models incorporate component representation, interface, interaction, assembly relationships, and embedded system features. All these models are based on the NIST core product model (CPM), a knowledge-oriented product information model developed to meet the requirements of next generation product development systems (Fenves 2001, Fenves et al. 2005).

[1] http://www.sysml.org/,
http://www.aitcnet.org/dodfw/

Table 1: Existing Models, Methodologies, Tools and Standards for Embedded Systems

Models
Models for Reactive Systems: finite state machines (FSM), Behavior FSM, Codesign FSM, State Charts, Program State Machines (PSM), Communicating Interacting Processes (CIP), Esterel, Specification Description Language (SDL), Synchronized Transitions, etc
Models for Transformational Systems: Synchronous Data Flow (SDF), etc
Control/Data Flow Models: Boolean Data Flow, Hierarchical Flow Graphs, Extended Syntax Graphs, Petri nets
General Purpose Languages: ANSI C, Occam, ADA (Program language), VHDL(Very high speed integrated circuit hardware description language), UML/SysML (Unified Modeling Language/System Modeling Language)
Methodologies
Heterogeneous System Design Methodologies: Mostly based on methodologies for ASIC design, such as, COSYMA (ANSI C), Olympus (Hardware C), Ptolemy (Stars, Galaxies), MOOSE (Model-based Object Oriented Systems Engineering), MDAF (model-driven architecture and framework), etc.
General Purpose Methodologies: Structured Analysis (Data Flow Diagram), Structured Development for Real-Time Systems, etc.
General Purpose OO Methodologies: Booch Method, ROOM (Real-time OO Modeling)
Tools
Generic Modeling Environment, Giotto, HyTech, Metropolis, Mocha, POLIS, Ptolemy II, etc.
Standards
STEP 10303-AP 233, SysML (System Modeling Language), DoDAF (DoD Architecture Framework)

3 Overview of Embedded System Design

Generally, the object-oriented system engineering method includes the following development activities:1) analyze needs, 2) define system requirements, 3) define logical architecture, 4) synthesize candidate allocated architectures, 5) optimize and evaluate alternatives, and 6) validate and verify the system. In this section, we provide an overview of embedded system design.

3.1 Component-based Design

As defined earlier, an embedded system is a specialized computer system that is part of a larger system or machine. Typically, an embedded system is housed on a single microprocessor board with the software (programs) stored in some form of read-only memory (e.g., 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.

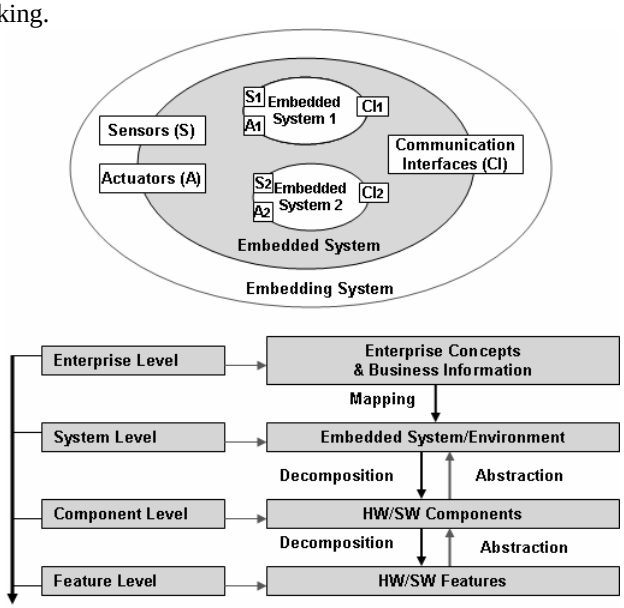

Figure 1: Four-level modeling for the life-cycle development of an embedded system

The modeling principle we adopted in this research identifies four abstraction levels for the design of an embedded system (Zha, Fenves and Sriram 2005a): 1) enterprise level; 2) system level; 3) component level; and 4) feature level. Fig.1 shows four-level modeling for the lifecycle development of an embedded system. Details are described as follows:

1) The enterprise level provides a unified view of the system and its environment by capturing enterprise-related concepts, including business process information.
2) 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.
3) The component level represents the system in terms of a set of composed components. A HW/SW component may be further decomposed into sub-components. A composite component is an aggregation 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.
4) The feature level defines the internal structure of simple HW/SW components. A component is structured using a set of related features, implemented in a feature description or a programming language. Thus, the design process of a component at the feature level corresponds to the feature-oriented design process similar to the traditional object-oriented process.

The basic concepts in the component-based modeling approach are systems, components, connectors, where a system is a configuration/assembly. Both components and connectors have connection points called ports for components and roles for connectors (Fig.2). Thus, design elements include components, connectors, ports, and roles. 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.

UML 2.0 includes a set of constructs about components and their assembly. The component description in UML 2.0 now can include a set of ports, a set of parts, a set of connectors and behaviors. Thus, in UML 2.0, major improvements have been added to support component-based modeling.

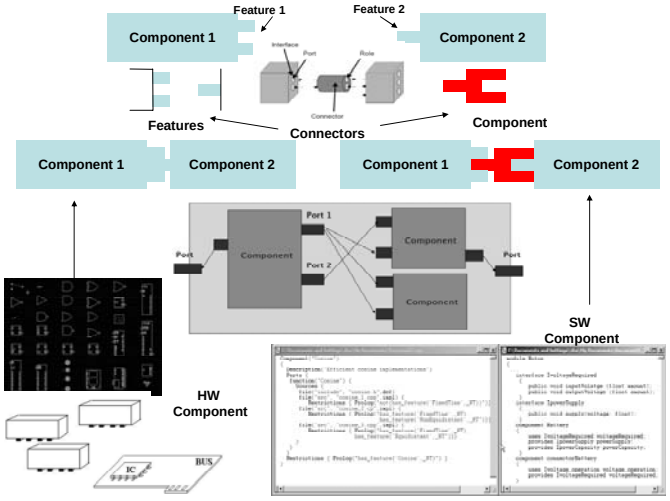

Figure 2: Embedded system component-connector model

3.2 Hardware and Software Co-Design

The co-design of hardware and software is the most critical but difficult issue in embedded system design. The hardware/software co-design decisions can profoundly impact the quality and cost of embedded systems and products. The co-design process of HW/SW starts with an architecture independent description of the system functionality, the analysis of constraints and requirements on the system, and the statement of objectives. This description is independent of HW and SW, and several system representations may be utilized, e.g., finite state machines (FSM). The system is then described by means of a programming language, which is next compiled into an

internal representation such as a data control flow description. This description serves as a unified system representation that can represent HW or SW. HW/SW functional partitioning is performed on this unified representation. After this step has been completed, HW, SW, and related interfaces are synthesized. Evaluation is then performed. The partitioning process is iterative, and if the evaluation does not meet required objectives, another HW/SW partition is generated and evaluated. Note that the partitioning stage and the integration phases are common to all codesign methodologies. Codesign is still a relatively new, changing approach, so there is not one set standard for how it must be done. Many variations exist.

3.3 Platform-based Design

Product family is a group of related products that share common features, components, and subsystems, and satisfy a variety of market niches. Product platform is a set of parts, subsystems, interfaces, and manufacturing processes that are shared among a set of products. A product family comprises a set of variables, features, or components that remain constant in a product platform and from product to product. Developing a family of systems instead of a series of separate systems offers many economic advantages. Gathering information about common features/properties of the systems enables the development of common, reusable core assets of the family. Information about differences between the family members can be used for the development of an extensible architecture of the common core assets. Here, two processes may be involved: 1) platform/ family construction; and 2) platform/family evolution. Platform-based design (PBD) aims to reduce the complexity of design task and design reuse. In practice, a set of components, which is used to build a system, is strongly related to its application domain. Thus, by establishing a well-defined set of components, a platform, and by validating it in a particular application type, it could be reused in future designs within such domain. By relying on already developed and validated platforms, the partitioning step can be done more easily, by mapping automatically the system functionality to the platform modules. PBD requires a "standard" architecture, to which components are interfaced, and to which a wrapper can be generated and PBD often uses interfaces. Generally, a PBD provides a generalized structure (product/system family architecture, PFA/SFA) to pure component-based design by providing architectural constraints on embedded system implementations. Today, platform-based product/system family design has been recognized as an efficient and effective means to realize sufficient product/system variety to satisfy a range of customer demands in support for mass customization.

4 UML Representation for Embedded System Model

4.1 Overview

An Open Embedded System Model (OESM) is developed to provide a standard representation and exchange protocol for embedded systems and system-level design, simulation, and testing information. In this section, we discuss in detail the various components of OESM, related to models of embedded system artifacts, embedded system components, embedded system features, embedded system configuration/assembly, embedded system family, and design rationale. Representations of a set of features may be entity relationship, dataflow, object communication, assembly, classification, stimulus-response, and state transition diagrams. The purpose of the multiple representations or views is to add flexibility in responding to evolving design paradigms, life cycle models, etc. We use UML notation and diagrams to explain the embedded system models.

The NIST Core Product Model (CPM) provides the basic foundations within function-behavior-structure (form) framework for the next generation CAD systems. However, CPM currently focuses mainly on 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). In this research, CPM is extended for modeling embedded systems. Specifically, a complete information model is defined, which consists of customer requirements, design specifications, HW/SW components (artifacts, functions-behaviors, geometry and material for HW, architecture, and code for SW), HW/SW interfaces, ports, etc., as follows:

Embedded System Model {
    Requirements;
    Specifications;
    ES (HW/SW) Artifacts;
    ES (HW/SW) Features;
    ES (HW/SW) Functions-Behaviors-Forms;
    ES (HW/SW) Performance Objectives and Constraints;

Relationships;
Platform/Family;
Design rationale; }

4.2 Main Schema

Fig.3 shows the main schema of OESM. The schema model incorporates information about design specification, partitioning, embedded system specification, and HW/SW component composition (part-of); and configuration/assembly relationships, embedded system platform/family, and design rationale.

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 **ESComponent** class in the **ESComponent** package, whether a HW/SW sub-system or component, is made up of one or more HW/SW features, represented in the model by **ESFeature** class in the **ESFeature** package. The **EmbeddedSystem** and **ESComponent** classes are subclasses of the **ESArtifact** class (extended from NIST-CPM **Artifact** class). **ESFeature** 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 **ESConnector** class in the **ESConnector** package. Connectors may be either features or components or subsystems composed by features or components. Each feature may have a list of tradeoffs and rationale associated with it. In the feature-based design method (below), a feature is any distinctive or unusual aspect of a system, or a manifestation of a key engineering decision.

Specifically, **ESArtifact** refers to an embedded system or one of its hardware/software (HW/SW) components. **ESArtifact** is specialized into two classes: **HWArtifact** and **SWArtifact**. **HWArtifact** refers to a hardware system/component in an embedded system, which is an aggregation of **HWFunction**, **HWForm** and **HWBehavior**. **HWFunction** represents what the artifact is supposed to do; **HWForm** represents the proposed design solution for the design problem specified by the hardware function; and **HWBehavior** represents how the hardware artifact realizes its function. **HWForm** itself is the aggregation of **Geometry**, the spatial description of the artifact, and **Material**, the internal composition of the hardware artifact. **HWFeature** represents any information in the **HWArtifact** that is an aggregation of **HWFunction** and **HWForm**. **SWArtifact** refers to a software system in the embedded system or one of its software components, i.e., which is an aggregation of **SWFunction**, **SWForm** and **SWBehavior**. **SWFunction** represents what the software artifact is supposed to do; **SWForm** represents the proposed solution for the design problem specified by the software function; **SWBehavior** represents how the software artifact implements its function. **SWForm** 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**. **SWFeature** represents any information in the **SWArtifact** that is an aggregation of **SWFunction** and **SWForm**. All the above entities have their own independent containment ("part-of") hierarchies.

For more details of all other related packages (system specification, requirement, system partitioning, view, embedded system component, embedded system connector, composition association, design rationale, embedded system platform/family, etc.), the reader can refer to (Zha, Fenves and Sriram 2005a, 2006).

5 Feature Semantic Models for Embedded Systems

As mentioned above, assembly/composition relations of an embedded system can be extracted based on the connecting HW/SW features (e.g., mating and joining features for HW) after specifying connecting relationships and methods between HW/SW components. In this section, we discuss in detail the semantic feature models for embedded systems, i.e., the open embedded system feature model (OESFM), related to models of embedded system artifacts, embedded system components, embedded system features, embedded system configuration/assembly, platform/family, and design rationale. In nature, OESFM can be considered as a feature view (like functional view, structural view, technology view, behavioral view) of OESM as shown in Fig.3, the main schema of the OESFM. In addition, feature models are widely used to capture common and variant concepts among systems in a particular domain. However, the lack of a formal semantics of feature models has hindered the development of this area. We further develop a semantic web environment for modeling and verifying feature models using ontologies.

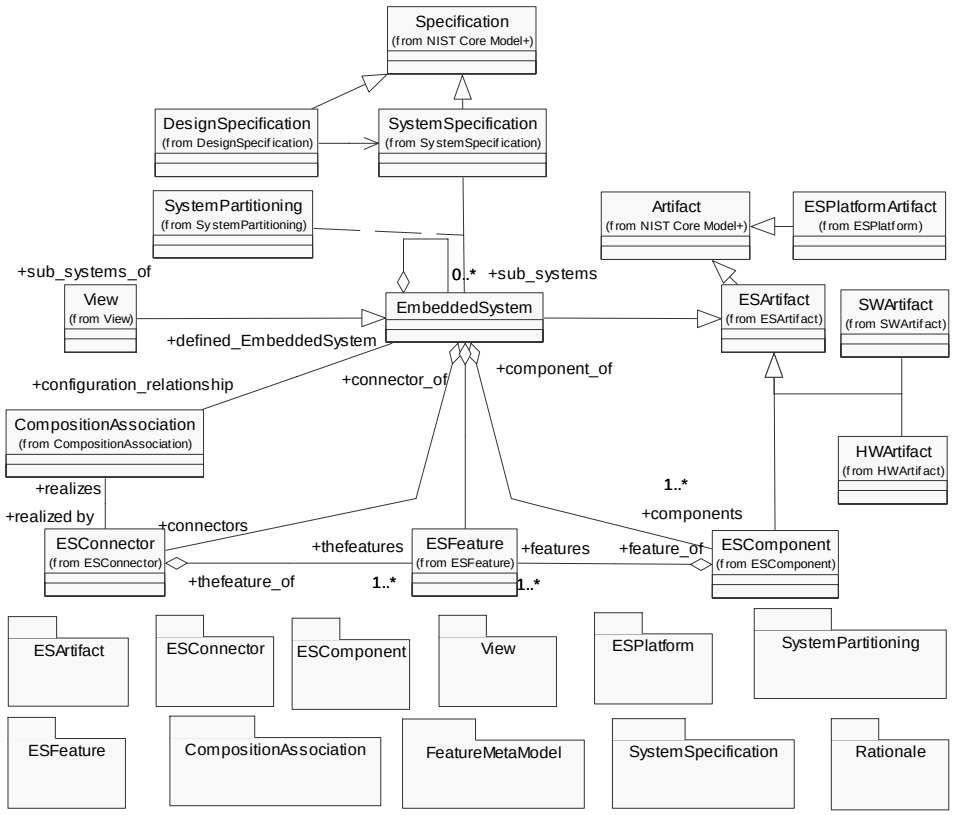

Figure 3: Main schema of the Open Embedded System Model

5.1 Feature Semantic Modeling

5.1.1 Feature Definitions

A feature is any distinctive or unusual aspect of a system, or a manifestation of a key engineering decision. There are four categories of features (Riebisch 2003, Riebisch et al. 2004) as defined in (Zha, Fenves and Sriram 2005a):

(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 be expressed through use cases, scenarios or structures.

(2) Structural features including form features and interface features express 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 value or assignment of quality, e.g., color.

(4) Concept features represent an additional category for structuring a feature model. They encapsulate abstract features within a hierarchical feature structure. The root of the hierarchy always represents a concept feature. Features in this category have no concrete implementation, but each of their sub-features provides one. The feature “electronic current protection” represents an example for such a feature.

5.1.2 Feature Diagrams and Feature Relations

A feature diagram provides a graphical notation that shows the hierarchical organization of features. The root of the tree represents a concept (artifact, component) node. All other nodes represent different features. Features can also be decomposed in AND-(EX)OR decomposition trees. An AND decomposition of a node in the feature space means that all its constituents must be available; an OR requires an arbitrary (>=0) number of constituents, and an

EXOR requires precisely one constituent. Assume that the concept *C* be selected, the following definitions can be derived on its child features (Czarnecki and Eisenecker 2000): 1) mandatory – The feature must be included into the description of a concept instance, 2) optional – The feature may or may not be included into the description of a concept instance; 3) alternative – Exactly one feature from a set of features can be included into the description of a concept instance; and 4) OR – One or more features from a set of features can be included into the description of a concept instance. In addition, a feature diagram itself cannot capture all the inter-dependencies among features, for example, two relations among features (Wang et al. 2005, Sun et al. 2005): 1) requires: The presence of some features in a configuration requires the presence of some other features; 2) excludes: The presence of some features excludes the presence of some other features. These relations are usually presented as additional constraints in a textual description.

Here, we extend the above definitions of feature relations. The extended assembly/composition relations are shown in Fig.4: 1) belong-to or inclusion relations (lines with arrowheads), 2) inter-feature relations (green line with solid roundheads), and 3) assembly/joining relations (ports) (red line with solid roundheads). In other words, each component has been or is to be completely designed with its associated features, and functional relationships are built between those component features by attaching some linkages/connections (relations). Therefore, a generic assembly liaison graph (GALG) can be formed and used to represent the relations on the entire composite structure (embedded system). Specifically, a GALG is designated to represent graphically the linkages between features and features to components. It can also represent assembly hierarchy and the connectivity of the whole system (components to components), based on these inter-feature relations and assembly/joining relations.

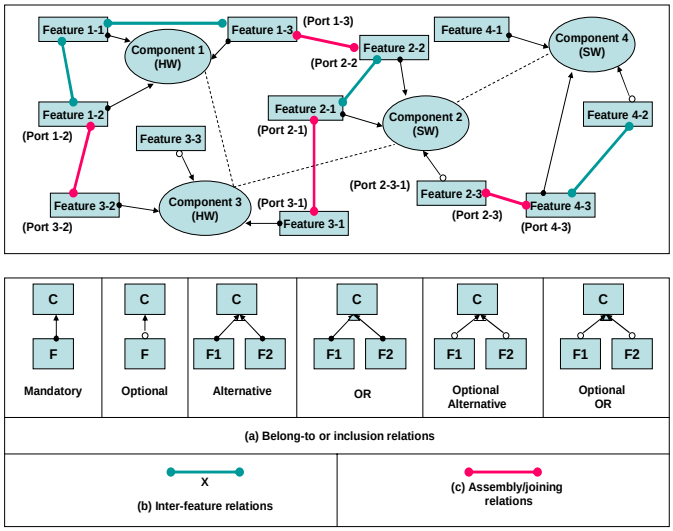

Figure 4: Assembly/composition relations among HW/SW components

5.2 Feature-Based Component Modeling for Embedded Systems

5.2.1 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 categories of features compose both hardware features and software features, that is, the hardware feature may be specialized into HW concept feature, HW function (behavioral) feature, HW parameter feature and HW interface feature; similarly, the software feature generalizes SW concept feature, SW function (behavioral) feature, SW parameter feature and SW interface feature. Normally, interface features are also called ports, thus, we may have a SW port and a HW port, accordingly in the software and hardware features. A SW port is specialized into Input Port (Requested Port), Output Port (Provided Port), In-Out Port (Resource Port, and Configuration Constants); A HW Port is specialized into Input Port (Source Port Requested Port), Output Port (Destination Port, Provided Port), etc.

5.2.2 Embedded System Feature Interactions

Embedded system connectors realize 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. Embedded system connectors can be specialized into subclasses: hardware connectors, software connectors and hardware-software connectors. Hardware connectors realize connections between hardware components or subsystems in the embedded system. Software connectors realize connections between software components or subsystems in the embedded system. Hardware-software connectors realize connections between hardware and software components or subsystems in the embedded system. Differing from interface features of HW/SW components, interface features of connectors are sometimes called roles.

The scenario of feature interactions in an embedded system can be described as: HW-HW, SW-SW, and HW-SW (Zha, Fenves and Sriram 2005a). We propose an approach for modeling feature (port/role) interactions to comply with the component-connector model based on UML 2.0. We also model feature interactions with a feature-solution graph. This graph serves two purposes. First, it can be used to pinpoint feature interactions. Second, it can guide as 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.

5.2.3 Embedded System Feature Model Representation

Feature modeling is an activity of creating 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 for each relationship among these features. Feature modeling is usually based on a two-level structure: 1) a meta-modeling level, which defines the types of features that can be used, their properties, and their mutual relationships; and 2) an entity modeling level where the feature model for the entities of interest is constructed in terms of the meta-model. 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 (Beuche 2003).

We use the UML-based formalisms to represent the feature meta-model (Zha and Sriram 2004). The basic ideas can be summarized as follows. A feature can have sub-features, but a group mediates the connections between a feature and its sub-features. 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.

5.3 Embedded System Feature Semantic Model with Protégé-OWL

We use Protégé-OWL to precisely capture the relationships among features in feature diagrams and configurations (Fig.5). OWL reasoning engines such as RACER are used to check for the inconsistencies of feature configurations automatically. As part of the environment, we also integrate a feature-modeling tool (e.g., CaptureFeature) to facilitate the visual development, interchange and reasoning of feature diagrams represented as ontologies.

6 Platform-based Design by Feature

Product family/line members share common features and also differ in certain features. Therefore, a platform could be generated based on features and reused in future designs. A set of components/features used to build a system is strongly related to its application domain. Platform-based design by feature (PBDF) aims to reduce the complexity of design task and design reuse based on features. For instance, to effectively implement platform-based SoC design, designers could use standard architectures such as the interface unit as the CoreConnect bus or some other industry standard bus like an AMBA bus. Channels are used as connectors, and the channel refinement is used to attach properties to the connector in order to pick an interface implementation from a library, which is compatible with the standard bus. Generally, the bus interfacing or the wrapper generation is more complicated in general component frameworks than in platform frameworks due to the absence of interface standards and connectors.

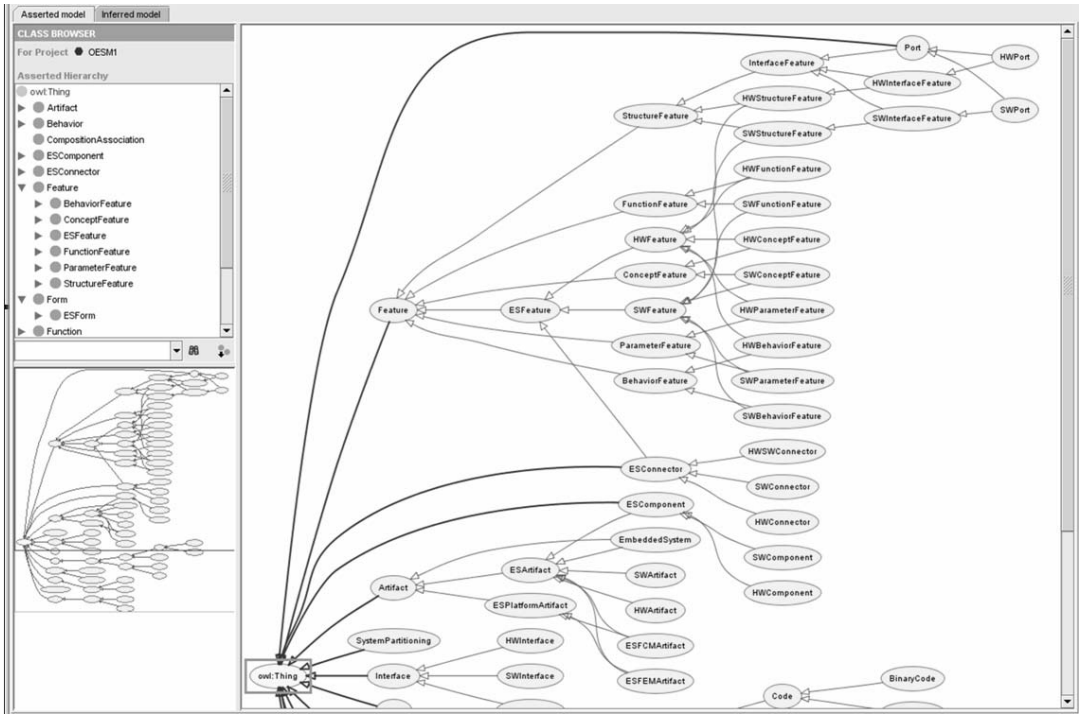

Figure 5: Embedded System Feature Semantic Model with Protégé-OWL

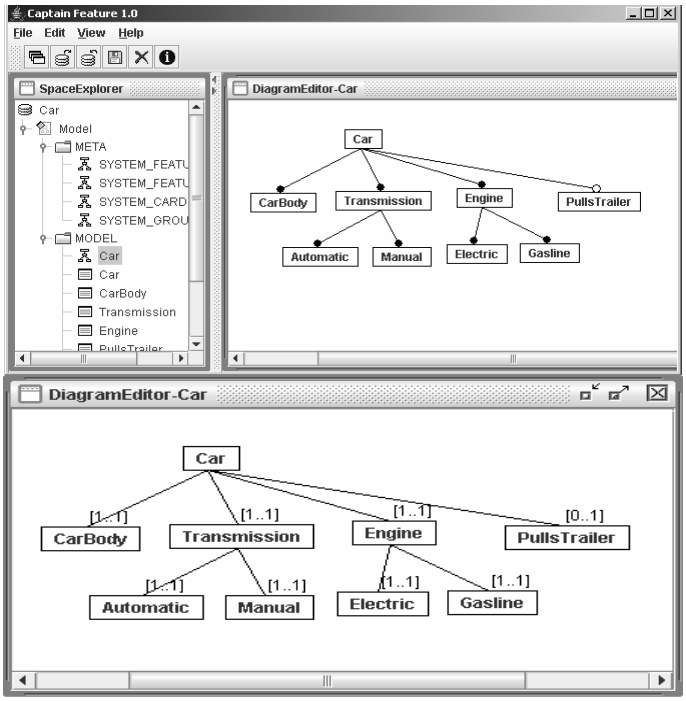

Figure 6: Car Feature model

A feature model- a hierarchical structure to the features can be used to describe the commonalities and variability/differences between the individual hardware/software systems. Commonalities can be modeled by common features (mandatory features whose ancestors are also mandatory), and variablities can be modeled by variant features, such as optional, alternative, and or-features. For instance, in the typical Car feature model (Fig.6), we can see that (Car, CarBody, Transmission, Manual, Engine, Gasline) and (Car, CarBody, Transmission,

Automatic, Engine, Electric, Gasline, PullsTrailer) are possible family configurations derived from the Car feature model. However, not all combinations of features are valid. For example, the configuration (Car, CarBody, Transmission, Automatic, Manual, Engine, Gasline) is invalid since the features Automatic and Manual are exclusive to each other.

Based on the feature-based component model above, we define an embedded system platform artifact (Fig.7) represented by **ESPlatformArtifact** class in the **ESPlatform** package (hardware platform, software platform, system platform). **ESPlatformArtifact** is subclass of **Artifact** class in the NIST CPM package. **ESFCMArtifact** (in the embedded system platform/family construction package **ESFCM**) and **ESFEMArtifact** (in the embedded system platform/family evolution package **ESFEM**) are subclasses of **ESPlatformArtifact** and **ESArtifact**, representing embedded system platform (family) to be constructed and evolved. These packages can support family-based design for mass customization, including the embedded system family construction model (**ESFCM**) and the embedded system family evolution model (**ESFEM**).

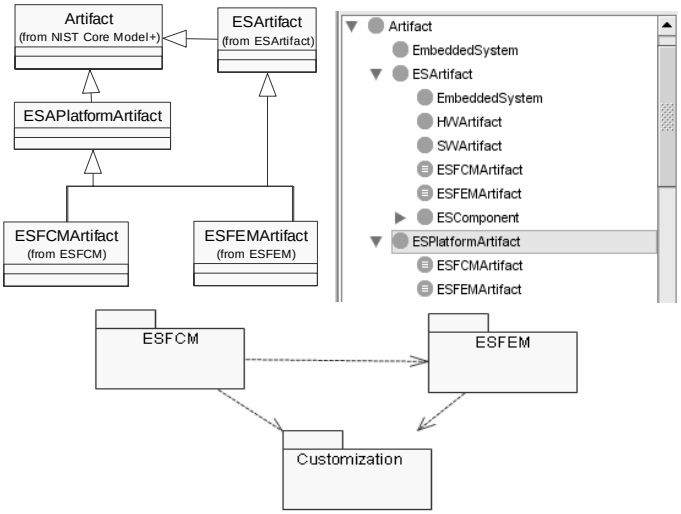

Figure 7: Class diagram and package of embedded system platform/family and customization

7 Feature-Based Hardware/Software Co-Design

We propose a novel feature-based approach to the co-design of hardware (HW) and software (SW) in embedded systems. The feature-based model serves as the basis for developing reusable and adaptable components/artifacts. The underlying SW and HW components are determined through feature configuration, and thus HW/SW co-design is implemented by using feature-component mapping and component generation, which may be associated with feature creation, configuration, analysis and reuse. Details are described in (Zha, Fenves and Sriram 2005a).

7.1 Feature-Component Mapping

During the form 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 the component design process. The implementation of 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 one 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. The feature configuration determines the underlying SW and HW component configuration.

7.2 Component Generation

We developed an approach to generating system components from functional as well as non-functional requirements. The generation approach is based on two processes: 1) Feature-form (FrFm) graphs, 2) Top-down component composition. The FrFm-graph captures architectural knowledge in the form of desired features (e.g., functional and non-functional requirements) and forms representing solutions that realize these features (e.g., architectural and design patterns). The steps in this process are: i) derivation of a reference architecture that meets the set of functional requirements; ii) application of known design solutions/forms focusing on non-functional requirements as codified in the FrFm-graph. Typically, the generation technique requires several iterations. These iterations might also involve backtracking steps because we usually have to deal with conflicting requirements. Basically, there are two spaces, namely the feature space and the form/solution space, recognized in the FrFm-graph. The Feature (Fr) space contains the requirements, whereas the Form (Fm) space contains forms/solutions addressing these requirements. As described in Section 5.1, features as well as forms are decomposed in AND-(EX)OR decomposition trees. An AND decomposition of a node in either the feature or the form space means that all its constituents must be available, an OR requires an arbitrary (>=0) number of constituents, and an EXOR requires precisely one constituent. Here, the key idea is that a feature in the Feature Space may select a form in the Form space as defined by directed selection links between nodes. It is also possible to explicitly rule out a particular form, which is done by connecting a feature to a form with a negative selection link.

8 Feature-based Embedded System Design Rationale

Rationale is an explanation of the fundamental reasons, in particular an explanation for the working principles of some device in terms of laws of nature. "Design Rationales include not only the reasons behind a design decision but also the justification for it, the other alternatives considered the tradeoffs evaluated and the argumentation that lead to the decision." This model and approach were successfully used in the NIST Design Repository Project (Szykman et al. 1999). Although design rationale model and its capture approach have been investigated for a decade, there is still no effective definition and capture approach. Design Rationale can be classified into static design rationale and evolutionary design rationale (Wang et al. 2003). The evolutionary design rationale can be retrieved from the static design rationale if using an appropriate approach/algorithm. Elements of design rationale are: a) Issues (what to decide?), b) Criteria (what is the basis for the evaluation?), c) Alternatives (what are the possibilities?), d) Justification (what reason?), e) Evaluation (how well the alternatives match the criteria?), f) Decision (which alternative do we select?), and g) Respondent (who's responsible for the decision?).

Here, we extend NIST Design Rationale Models (Szykman et al. 1999, Wang et al. 2003) for feature-based embedded system design. Embedded system design rationale could be generated at any stage in the embedded system (HW/SW) development process: requirement, analysis, design, implementation, and maintenance. Here, we propose a feature-based design rationale modeling method to provide mechanisms for capturing and tracking each feature of the embedded system. In the feature-based design rationale capture method, a feature is any distinctive or unusual aspect of a system, or a reflection of a key engineering decision. The features in an embedded system can differentiate other systems in the application domain. Each feature can have a list of tradeoffs and rationale associated with it. Therefore, a new way of looking at a system can be represented by adding a new view of features of the system. The feature-based design rationale modeling method is advantageous over other methods. This approach makes the embedded system design and development process become a process of answering questions about the features of a system rather than a cookbook-like procedure defined by a particular development method (Bailin 1990).

Fig.8 shows a partial feature-based design rationale model. The classes **FeatureDesignRationale** and **FeatureEvolutionRationale** are subclasses of **Rationale**. Design justification is the principal attribute of **Rationale**, defined by **FeatureDesignJustification**. The feature design justification may be domain feature technology (e.g., domain-specific methods, recommendation, standard, law.) or implementation/development specification (e.g., design decisions, requirements or regulations, implementation decisions, architecture style, process coordination method, implementation method, communication protocol, computational algorithm, implementation method), represented by **Technology** and **DevelopmentSpecification**, respectively. There are two kinds of evolution rationales (Wang et al. 2003): family derivation rationale and design evolution rationale, represented by **FeatureFamilyDerivationRationale** and **FeatureDesign EvolutionRationale**, respectively. **FeatureDesign EvolutionRationale** records the reasons for design changes between a series or version and its predecessor(s). **FeatureFamilyDerivationRationale** captures the driving features/factors for these changes. The features/factors

driving evolution are defined by **FeatureEvolutionJustification**, two kinds of which are the domain technology feature evolution and development specification evolution, defined by **TechnologyEvolution** and **DevelopmentSpecification Evolution**, respectively.

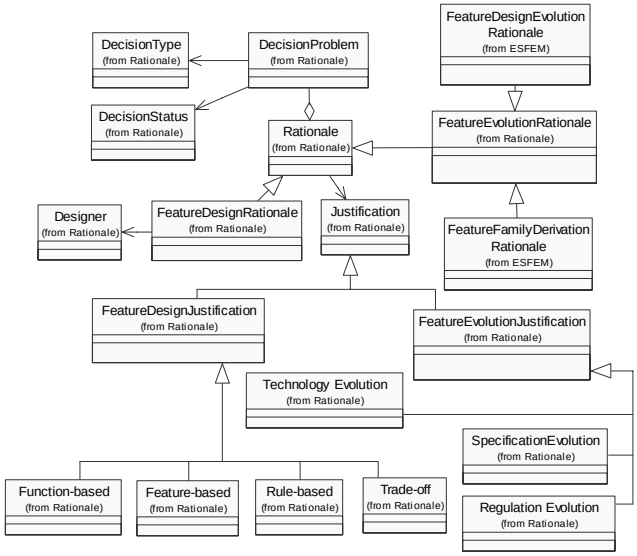

Figure 8: Feature-based design rationale model (partial)

9 Feature-based Collaborative Embedded Systems Design Framework

This section explains how the developed feature semantic model above provides a feature-based HW/SW component co-design framework allowing the designer to develop collaboratively a virtual embedded system prototype through assembling virtual components. Based on the component-based approach, a unified design-with-modules scheme is adopted to model the embedded system design process and a web-based knowledge-intensive distributed module modeling and evaluation (KS-DMME) framework is developed as a collaborative cyber-infrastructure to support CORBA-based distributed network-centric embedded system design (Zha and Sriram 2005b). The framework intends to link distributed, heterogeneous HW/SW (design) modules and tools and assist designers in evaluating design alternatives, visualizing trade-offs, finding optimal solutions, and making decisions on the web. It also enables designers to build integrated design models using both the local and distributed resources (e.g., local and distributed HW/SW modules) and to cooperate by exchanging services. The client (browser) / knowledge server architecture allows embedded system design modules to be published and connected over the web to form an integrated intelligent models/modules network.

Currently, we are working on a service–oriented architecture (SOA) and framework for collaborative embedded system design. Consider a typical scenario, where a system designer/integrator, such as an auto manufacturer, outsources the design and manufacturing of (HW/SW) sub-systems such as car antilock braking systems (ABS) from different vendors. If the auto manufacturer wants to design a complete ABS with subsystems (traction detector, brake modulator, hydraulic systems, microcomputer controller, etc.) designed by different vendors, the vendors provide an XML/OWL(Extensible Markup Language/Web Ontology Language) formatted OESFM and CAD model, of the subsystems to the auto manufacturer instead of sending the entire CAD model. The system designer/integrator can easily generate GALGs (generic assembly liaison graph) from the OESFM model. The GALGs are linked to the corresponding design models of subsystems or components. These design models in certain proprietary CAD formats are translated into CAD kernel formats, as soon as the vendors send the OESFM model to the system designer/integrator. The system designer/integrator can obtain the design models in the CAD kernel format when he/she sends a request for viewing a specific component from the GALG tool. The auto-manufacturer therefore can decide the assembly HW/SW components to be connected or joined. The determined assembly components are loaded into the configuration engine (configurator) and the system designer/integrator can specify a connecting/joining method between HW/SW components. The new OESFM model and its corresponding GALG are generated automatically based upon the OESFM formalism. During the collaborative embedded systems design, design participants typically use different HW/SW CAD systems. To generate a complete assembly, each design

model needs to be translated into a single CAD format, which can be accomplished by using specialized translators (Kim et al. 2004). Our solution for this is to use an XML/OWL-based modeler (an OWL compatible modeler is being developed) that has the ability to process assembly or embedded system ontologies to improve interoperability.

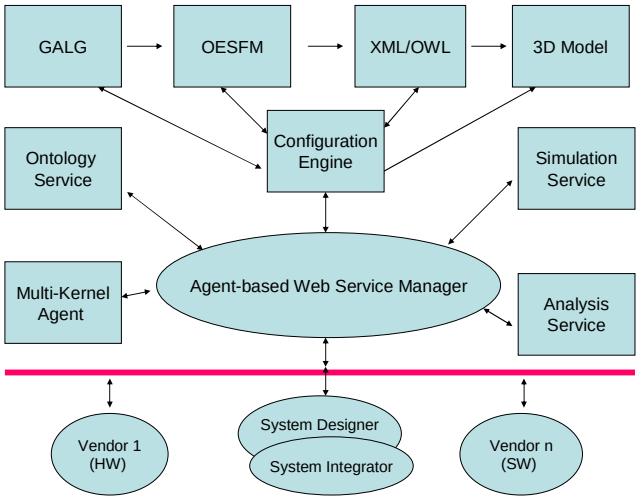

Figure 9: Service-oriented collaborative embedded system design

Fig.9 shows how collaborators (system designer/integrator, vendors, manufacturer, etc.) can share OESFM models interacting with different CAD systems in a service-oriented collaborative embedded system design environment. Consider a system designer/integrator who wants to assemble/configure two HW/SW components designed by vendor 1 and vendor 2. Through the SOA, the system designer/integrator can obtain OESFM models of HW/SW components remotely and generate an assembly (embedded system) without receiving the entire CAD model. Detailed processes are described in the steps below.

1. A system designer/integrator analyzes design requirements, specifications, and system partitions and defines/synthesizes allocated architectures.
2. The system designer/integrator requests OESFM models of sub-systems/ (HW/SW) components interested.
3. Vendors provide the requested OESFM models in XML/OWL formats to the system designer or integrator. The corresponding HW/SW CAD models are translated to the CAD kernel model and stored in the local database of each vendor. A third-party multi-kernel agent can be employed if the vendor doesn't have the capability to translate the CAD model to the kernel model.
4. The system designer/integrator reviews sub-systems/components with the aid of the GALG tool and a product/system viewer. Based on the system design' needs, he/she can then use the GALG tool to retrieve necessary HW/SW components in the kernel format from the vendors' database.
5. Once HW/SW sub-systems/components are selected, the system designer/integrator can load kernel models of the elected individual HW/SW components into the design engine and specify connecting/joining methods between the HW/SW components.
6. An OESFM model for the new embedded system is generated based upon the OESFM formalism. The new OESFM model can be sent to the vendors to share the assembly (embedded system) information.
7. When additional OESFM information is needed, such as the physical effects or behaviors of connectors, the system designer/integrator can request relevant service using the OESFM models through the SOA.

10. Case Study

In this section, a hydraulic measurement and control system (HMCS) (Fig.10), which aims to be a testing/diagnosing station for car antilock braking systems (ABS), is used as a case study to illustrate the application of feature technology and ontologies in embedded system design. This example is inspired by a weather station system described in (Grades 2003). The system is based on a small experimental microcontroller (e.g., 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. A list of the components and component features of HMCS, i.e., the instances of embedded system components and

features is provided in (Zha, Fenves and Sriram 2005a). Fig.11 shows a feature model (diagram) of HMCS, its UML representation and configuration, and the feature-based HW/SW co-design process. Several input and output options have to be implemented in HMCS. Each of the sensors is optional; the output options are formatted or unformatted output for the LCD display and for the serial output. Another output option is the use of the predefined SLIP (Serial Line Internet Protocol), UDP (User Datagram Protocol) packets. The model is completely independent from the implementation platform (see below) and could be reused without any changes on top of any other operating systems. Only the mapping functionality has to be exchanged. Each of the different output devices (USB, Serial, Modem/Internet and LCD) has almost identical sub-features. This is because all devices can be used for the same functionality. All devices would have to support all output options. The output formatting has to be set separately for each output device.

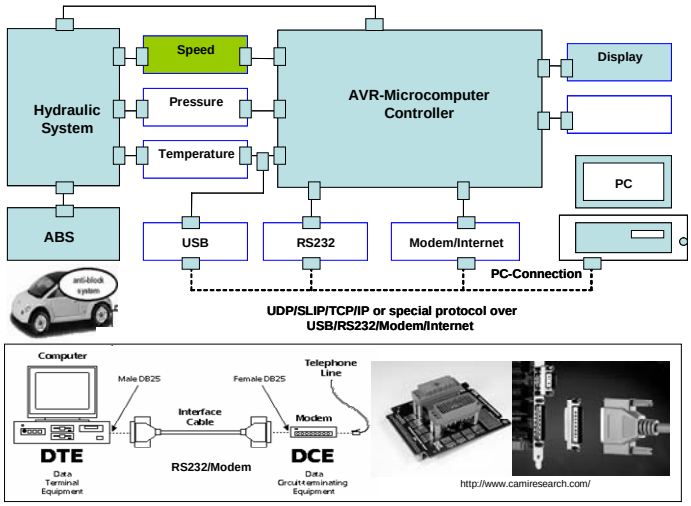

Figure 10: The hydraulic measure and control system (HMCS)

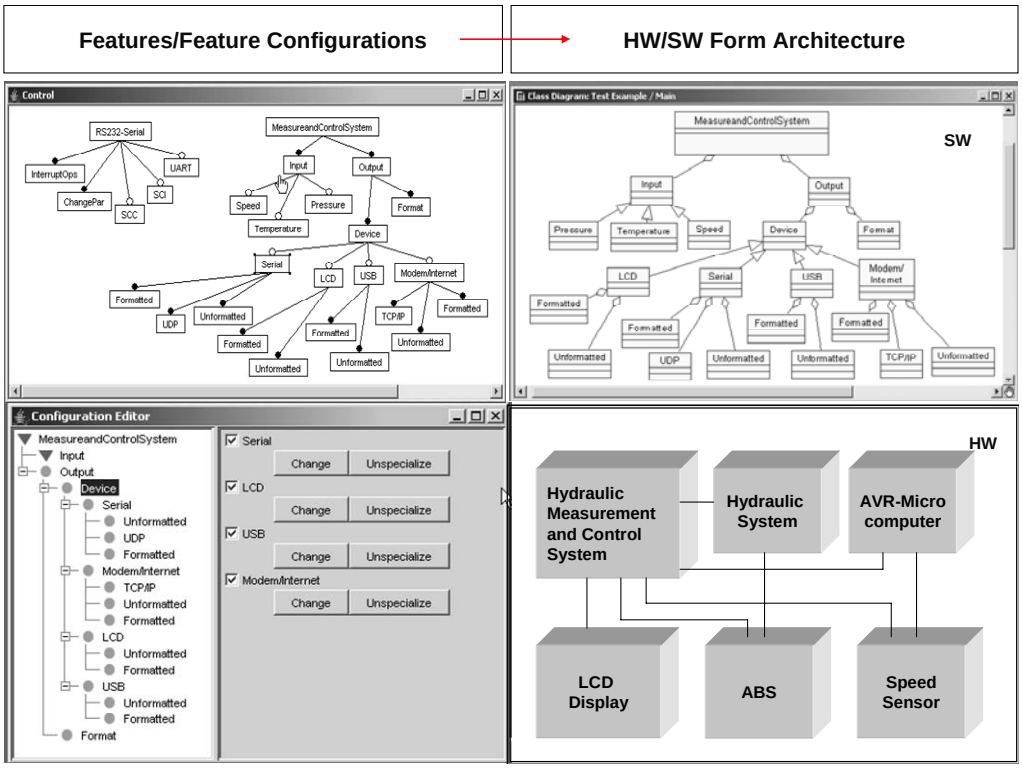

Figure 11: Components and feature diagram of HMCS and its feature-based HW/SW co-design process (Zha, Fenves and Sriram 2005a)

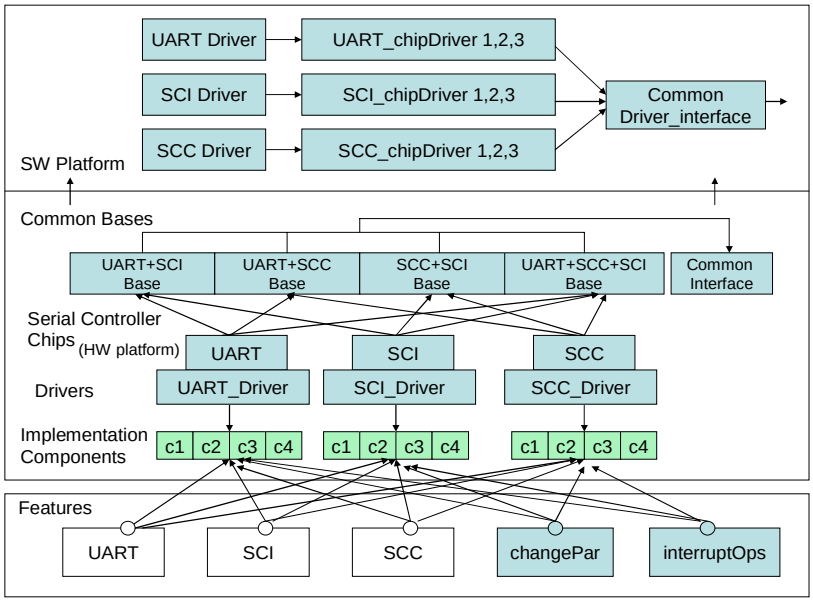

Figure 12: Platform configuration scenario for RS232 driver

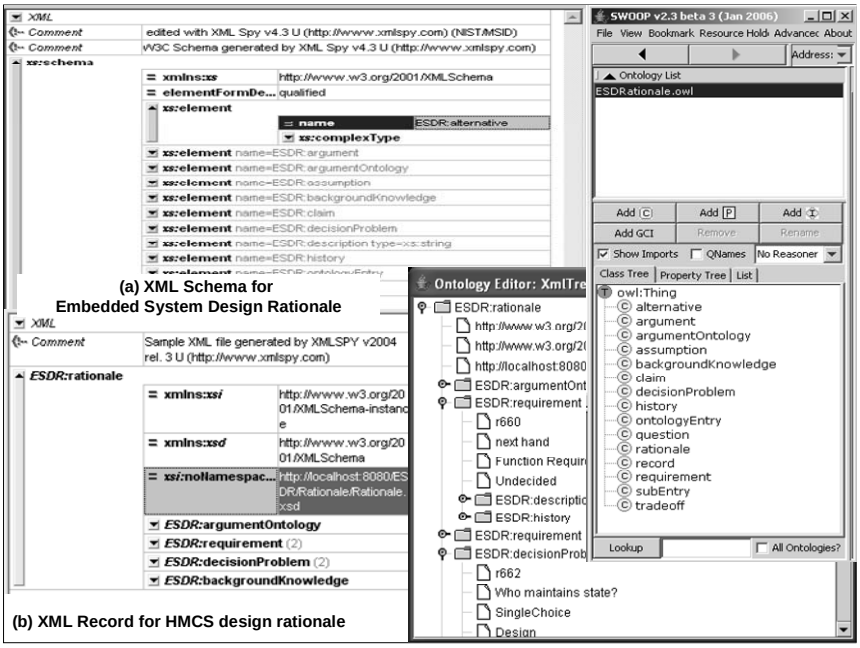

Figure 13: HMCS design rationale ontology

More specifically, for the feature model of the RS232 software component, RS232driver, it realizes device driver support for 3 different serial controller chips: UART, SCI, and SCC. This is based on the work in (Beuche 2001). The implementation consists of 3 sets of classes realizing the 3 different chip drivers and an optional abstract class which allows providing a common interface for all drivers. All together, there are at least 68 component implementations: 3 drivers with each having 4 different implementations and 4 different sets for the common base of drivers with a common interface: UART+SCI, UART+SCC, SCC+SCI, and UART+SCC+SCI. The feature model is quite simple and only allows specifying the required drivers and if changeable parameters (changePar) and/or interrupt based operation (interruptOps) are required. The feature model allows only 28 different feature selections.

If only one or all of the three chips are required to support the same feature selection out of changePar and interruptOps an optimal implementation component can be generated. Thus, the optimal component implementation could be generated only from the given feature model or a subset of the potential usage scenarios. Fig.12 gives a platform configuration scenario for RS232 driver. Fig.13 provides a screenshot of HMCS design rationale. The description of the feature design and evolution decision for HMCS includes: 1) the summary of the tradeoffs that were considered in making decision such as development and deployment cost, prototyping time, adaptability, scalability, etc. 2) the ultimate rationale for the decision.

The application case study tested and demonstrated that the feature technology and ontology could be used effectively for HMCS modeling and design, including the HMCS feature model, feature configuration, HW/SW co-design, platform/family and design rationale, etc. Note that the resulting models and ontologies and their applications are not fully illustrated here due to space limits.

11. Conclusions

This paper summarized a feature technology and ontology for embedded systems modeling and design. The approach defined an object-oriented representation for the embedded system model (OESM/OESFM), in particular the feature semantic models, including models of embedded system artifacts, components, features, and HW/SW configuration/assembly, platform/family, design rationale, etc. Feature semantic models and ontologies are developed with Protégé-OWL for platform-based design, HW/SW co-design, design rationale, and collaborative design. The feature model is used to provide a formal description of embedded systems and to formalize knowledge about its instantiation process and design rationale. Details of the feature semantic modeling and representation are discussed. The models can provide a feature-based HW/SW co-design framework and allow the designer to develop a virtual embedded system prototype through assembling virtual components on the platform. Thus, this work contributes to the framework modeling and feature modeling and ontology for the design of embedded systems.

Acknowledgement and Disclaimer

The authors appreciate the valuable comments and improvements suggested by Prof. Steven J. Fenves during the course of the project. The authors would like to thank Prof. Jami Shah for his valuable comments and suggestions.
No approval or endorsement is intended or implied of any commercial product, service or company used for demonstration purpose by the National Institute of Standards and Technology.

References

[1] Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., Vincentelli, A.S. (2003) Metropolis: An Integrated Electronic System Design Environment, Computer, IEEE Computer Society, pp. 45-52

[2] Bailin, S., et al. KAPTUR: Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationale, 95-104. Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference. Liverpool, NY, September 24-28, 1990. Rome, NY: Rome Air Development Center, 1990.

[3] Beuche, D., Papajewski, Holger, Schroder-Preikschat, W. (2004), Variability Management with Feature Models, Science of Computer Programming, Elsevier Publishers

[4] 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

[5] Beuche, D. (2003), Composition and Construction of Embedded Software Families, PhD Dissertation, der Otto-von-Guericke-Universität, Magdeburg, Germany

[6] Berg, K. Müller, J., Bishop, J. and van Zyl, J. (2004), The Use of Feature Modeling in Component Evolution, Technical Report, 2004

[7] Bruin, H. and Vliet, H. (2001), Feature and Feature Interaction Modeling with Feature-Solution Graphs, 2001

[8] Chen, R.,. M. Sgroi, G. Martin, L. Lavagno, A. Sangiovanni-Vincentelli, J. Rabaey, Embedded System Design Using UML and Platforms, Proceedings of Forum on Specification & Design Languages 2002 (FDL'02), Marseille, France, September 24-27, 2002

[9] Czarnecki, K. and Eisenecker, U. (2000), Generative Programming: Methods, Tools, and Applications. Addison-Wesley, MA, 2000.

[10] Deursen, A. and Klint, P. (2001), Domain-Specific Language Design Requires Feature Descriptions, CWI Report, SEN-R0126, ISSN 1386-369X

[11] Eggermont, L. D.J. (ed.) (2002), Embedded Systems Roadmap 2002, Vision on Technology for the Future of PROGRESS, 30 March

[12] Esser, R. (1996), An Object-oriented Petri Net Approach to Embedded System Design, PhD Dissertation, Swiss Federal Institute of Technology at Zurich, Zurich, Switzerland

[13] Fenves, S., Foufou, S., Bock, C., Bouilon, N., Sriram, R.D., (2005), CPM 2: A Revised Core Product Model for Representing Design Information, NISTIR 7185, NIST, Gaithersburg, MD.

[14] Grades, zur Erlangung des akademischen (2003), Composition and Construction of Embedded Software Families, PhD Dissertation der Otto-von-Guericke-Universität Magdeburg, Germany

[15] Hassani, M., A Component-based Methodology for Real-time Decision-making Embedded Systems, PhD Dissertation, University of Maryland, 2000

[16] Kim, K.Y., Wang, Yan, Muogboh, O.S., Nnaji, B.O., Design Formalism for Collaborative Assembly Design, Computer-aided Design, Vol. 36, pp. 849-871, 2004

[17] Lee, Edward A. (2003), Overview of the Ptolemy Project, Technical Memorandum No. UCB/ERL M03/25, University of California, Berkeley, CA, 94720, USA

[18] Rozenblit, J. and K. Buchenrieder (editors) (1994), Codesign Computer-Aided Software/Hardware Engineering, IEEE Press, Piscataway, NJ

[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] Shah, J. and Mantyla, M., Parametric and Feature-based CAD/CAM: Concepts, Techniques, and Applications, John Wiley and Sons, Inc., New York, 1995

[22] Sriram, R.D., 2002, Distributed and Integrated Collaborative Engineering Design, Sraven Publishers, Glenwood, MD 21738

[23] Szykman S., Sriram, R., Bochenek C. and Racz, J., The NIST Design Repository Project, Advances in Soft Computing - Engineering Design and Manufacturing, Roy, R., T. Furuhashi and P. K. Chawdhry (eds.), Springer-Verlag, London, 1999, July, 1998, pp.5-19.

[24] Sudarsan, R., Han, Y.H., Feng, S. , Roy, U., Wang, F., Sriram, R.D. , Lyons, Kevin , Object-Oriented Representation of Electro-Mechanical Assemblies Using UML, NISTIR 7057, 2003

[25] Sun, J., Zhang, H.Y., Li, Y.F., and Wang, H. (2005), Formal Semantics and Verification for Feature Modeling, 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), IEEE Press, Shanghai, China, June 2005.

[26] 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.

[27] Wang, H., Li, Y.F., Sun, J., Zhang, H.Y., Pan J. (2005), A Semantic Web Approach to Feature Modeling and Verification. Workshop on Semantic Web Enabled Software Engineering (SWESE), The 4th International Semantic Web Conference (ISWC05), Nov. 6th-10th , Galway, Ireland, 2005

[28] Wang, F.J., Fenves, S.J., Sudarsan, R., and Sriram, R.D., Towards Modeling the Evolution of Product Families, Proceedings of CIE'03, 2003 ASME Computers and Information in Engineering Conference, September 2-6, 2003, Chicago, IL, (2003)

[29] Zha, X.F. and Sriram, R.D. (2004), Feature-based Component Model for Design of Embedded Systems, in Intelligent Systems in Design and Manufacturing V, edited by B. Gopalakrishnan, Proceedings of SPIE Vol.5605 (SPIE, Bellingham, WA, 2004), pp. 226-237

[30] Zha, X.F., Fenves, S.J., and Sriram, R.D. (2005a), A Feature-based Approach to Embedded System Hardware and Software Co-design, Proceeding of 2005 ASME DETC, Paper No. DETC 2005-85582

[31] Zha, X.F. and Sriram, R.D. (2005b), Distributed Modeling and Framework for Collaborative Embedded System Design, Proceeding of 2005 ASME DETC, Paper No. DETC 2005-85608

[32] Zha, X.F., Fenves, S.J. and Sriram, R.D. (2006), Object Oriented Representation for Embedded System using UML, Working Report (1st Draft, 2nd draft in progress), National Institute of Standards and Technology, USA