INVITED PAPER

Integration of Collaborative Analyses for Development of Embedded Control Software

By Sangsoo Park, *Member IEEE*, Kang Geun Shin, *Fellow IEEE*, and Shige Wang, *Member IEEE*

ABSTRACT | Model-based methodologies have been widely used to handle the increasing demand for rapid development of high-quality, real-time embedded control software. A key challenge in such model-based design is integration of various "collaborative" analysis methods to support the automation of the design process. Traditional analysis methods developed for analyzing specific system properties, however, are not designed for such integration, and thus cannot ensure that the information for the analysis will be provided at the design stage where the information is needed. Moreover, many traditional analysis methods depend heavily on complete and accurate design models which can only be applied to post-design verification and are unavailable for automation of the design process involving an early design stage, where implementation details are unknown. This challenge can be met by integrating analysis methods with the design process. We have developed such a framework combined with software modeling, execution platform configuration, and run-time monitoring mechanisms to enable accurate assessment of embedded software quality at early design stages. We have implemented and demonstrated the framework with a toolkit, called AIRES, that integrates software models, a virtual execution platform, and timing and schedulability analysis methods.

KEYWORDS | Collaborative analysis; embedded control software; real-time system; timing constraint

I. INTRODUCTION

From aircraft to automobiles, embedded control systems are becoming omnipresent. An important subset of such systems is embedded control software (ECSW) that implements the controls of physical processes (e.g., from pressing brake pedal, to increasing the brake fluid pressure, to applying brake pads in automobiles). The physical processes imply that ECSW must meet critical functional and cross-cutting system requirements to guarantee the correct system behavior. Analyses must, therefore, be applied through the whole ECSW development process to guide the design and verify the preservation of system properties.

The current ECSW development process follows some form of the V-diagram, as shown Fig. 1. The stages at the left side of the process focus on the system design, with each stage refining the intermediate designs generated at its immediate previous stage by filling in more details for implementation. The stages at the right side focus on the integration and verification of the implementation. The stages at the same level on both sides of the process consider the same system scope and design granularity.

Such a multi-stage ECSW development process usually requires multiple engineering groups in different disciplines to collaborate to meet the constraints of all system aspects. The complexity of ECSW and its development process makes it difficult to apply the traditional development methods based on textual function specifications with coding rules, manual source code generation, and code-level optimization to generate satisfactory products. This complexity is due mainly to the rapidly-increasing size and diversity of embedded systems. For example, vehicle

---

Manuscript received December 9, 2009. This work was supported by the Korea Research Foundation Grant funded by the Korean Govenment (MOEHRD) (KRF-2006-214-D00127), the Ewha Womans University Research Grant of 2009, Hitachi America, and VaST Systems.

S. Park is with the Department of Computer Science and Engineering, College of Engineering, Ewha Womans University, 11-1 Daehyun-dong Seodaemun-gu, Seoul 120-750, Korea (e-mail: sangsoo.park@ewha.ac.kr).
K. G. Shin is with the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109 USA (e-mail: kgshin@eecs.umich.edu).
S. Wang is with General Motors R&D and Planning, Warren, MI 48090 USA (e-mail: shige.wang@gm.com).

Digital Object Identifier: 10.1109/JPROC.2009.2039741

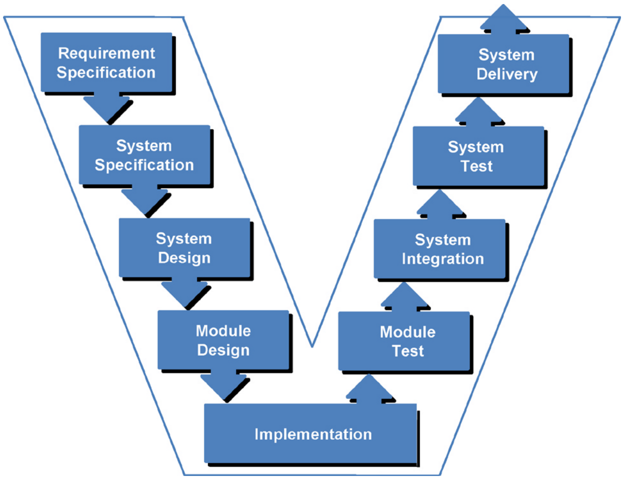


Fig. 1. ***V-diagram representing the ECSW development process.***

customers demand more functionality such as in-vehicle entertainment and convenience features; regulatory entities introduce new mandated regulations such as emission-control and fuel-efficiency. All of these will dramatically increase the number and types of control functions and devices in a vehicle system, which will, in turn, make the system complex. Constantly-evolving system specifications and components during the development result in design uncertainties, with which the information used to make the design choices and decisions may change until the end of the whole system development. Such uncertainties introduce another dimension of complexity, which is very difficult to manage to provide minimum but enough design flexibility to accommodate the uncertainties.

One promising technology to address the ECSW design challenges is the model-based method which has been adopted by many industries and organizations as a solution for development of large ECSW. The model-based method operates with models of systems and system components, either commercial-off-the-shelf (COTS) or created in-house, and reasons about their relationships and properties. With a model providing abstractions of behaviors and structures of the components, the functional specifications can be defined as an integration of the required executable models using model-based methods, thus hiding unnecessary implementation details and simplifying the design and system verification and validation. Source code will be automatically generated based on the models and the rules after an architectural optimization at the system-level. To this end, the development process can be considered as a sequence of transformations from an abstract model to the one with implementation details, during which analyses are performed to ensure that the transformed models will preserve critical properties of the system.

Another dimension to manage the system complexity is modularization with consideration of reuse. Numerous research efforts have been made to define modularized software with standardized infrastructures that allow flexible composition of software components from different developers/vendors. A key challenge in applying these research results is how to meet the non-functional requirements (related to timing, resource, etc.) that the ECSW development must account for, in addition to the meeting the usual functional requirements. These non-functional properties are critical, especially when ECSW is associated with physical processes. In particular, most physical processes are time-sensitive, and the correctness of their ECSW depends not only on the functional computation but also on the timeliness of the computation. This requires that the execution of ECSW must meet stringent timing constraints to satisfy the performance requirements of control functions when responding to stimuli from the external physical world. While the control algorithms designed by control engineers using existing technology typically assume zero computation delays only achievable with an unlimited amount of computing resources, the ECSW implementing these controls typically runs on a computing platform with limited resources for the reasons of size, cost, and power. Preserving system properties with the non-functional, sometimes conflicting, constraints requires analyses, given the fact that no theoretical foundation for composition of non-functional properties exists to date. In this paper, the word "analysis" is referred to as a mathematical process of examining and determining the system properties of interest with given input information. In other words, analysis methods and algorithms are the vehicles to perform the analyses. The multi-stage, multi-group ECSW development process makes it difficult to apply existing timing analysis methods due to the unavailability of accurate executable models in the early stages of ECSW development. Currently, developers rely on labor-intensive (thus expensive and time-consuming) simulations and prototyped systems to validate the correctness of an ECSW design. Simulations are usually ad-hoc and valid only for a particular configuration. Thus, they are not reusable across different products and configurations, and require additional resources and efforts for new simulations involving the evolving components even when minor design changes are made. Prototypes are useful for test-based verification and validation, which can be performed only after the implementation stage with complete knowledge of design details. As the cost of finding and correcting software errors, including timing errors, increases dramatically in the later stages of ECSW development, it is important to apply the analyses "collaboratively," starting from an early design stage in the process to detect and correct design errors resulting from the non-functional requirements. To make the analyses work collaboratively, we need support for mapping data representation in one format to another and feeding the results generated by one analysis to another.

To manage the complexity introduced by collaborative design and analysis across multi-stages, multi-groups, and

Table 1 Models and Languages for ECSW Development

Process	Models and Tools
Requirements	Natural language, use-case diagram, block diagram
Analysis	Differential equation, state-based formalism
Simulation	Continuous- and/or discrete-time control model
Design	UML (Unified Modeling Language)
Implementation	C/C++ code libraries

multi-disciplines, it is essential that the models and methods used for the ECSW development process must 1) represent an ECSW design with only the required information exposed to a group of people who may focus only on a single discipline and/or a system aspect, and 2) share a design among different groups and stages to allow collaboration. As a result, we need methods for identifying and assessing what information is needed at which stage, on which aspect, and for which group to maintain the leanness of the development process and, to avoid unexpected and undesirable complexity. Table 1 presents design models and modeling languages commonly used in the current ECSW development process.

To address the challenge of collaborative analyses throughout the development process, and ultimately enable rapid ECSW development, we need a framework that facilitates the integration of domain-specific modeling and multi-discipline analysis algorithms. To be concrete, we describe such a framework that has been developed and implemented in the *Automatic Integration of Reusable Real-time Embedded Software* (AIRES) toolkit at the University of Michigan. The framework supports the creation of integrated domain-specific models used at different design stages, provides mechanisms to interface with other tools to collect the required information, and enables the analysis algorithms used at different stages to work collaboratively for design refinements. The consistency between the original and refined models after applying the analyses is preserved by the constraints defined in the integrated metamodel that covers the shared concepts used for the collaborative analyses and the relations of these concepts. In this paper, we assume that all participating analysis algorithms preserve the system properties if they make changes to the model. Constraints in the metamodel and the property-preserving analysis together ensure the model consistency. As an implementation, the current AIRES toolkit integrates the modeling for application software, for a runtime architecture, for computing platforms, and for non-functional timing constraints. New methods that gather runtime information from a virtual execution platform and use the information in the analyses for design verification and refinement at an early development stage have been implemented and integrated in the AIRES toolkit. The framework with all of the developed techniques together enables the rapid development of ECSW with the current development process.

The rest of this paper is organized as follows. Section II discusses the background and the related work on ECSW development. Section III presents the system model and assumptions along with the framework architecture. Section IV details the integration of timing and schedulability analysis methods using the developed framework. Section V describes an example of Electrical Throttle Control (ETC) software development using the AIRES toolkit. The paper concludes with Section VI.

II. BACKGROUND

At every stage of real-time ECSW development, a design analysis is required to derive key parameters for implementation and check if the required system properties are preserved. The timing analysis, which affects the decisions on system scheduling, resource utilization, and application performance, is one of critical analyses for the timing property of real-time ECSW. For ECSW, different timing-analysis algorithms may be used to address the issues at different design stages. Multiple analysis algorithms may also be used collaboratively at the same design stage to obtain better results. Integration of these collaborative analyses to meet system-level design objectives as well as reduce the design complexity of each stage is, albeit difficult, highly desired.

Traditionally, the analysis at each stage is done independently. Some common examples include the performance analysis based on queueing theory, software task response time analysis, and system schedulability analysis. Although some of these analyses are based on mathematical underpinnings, simulation and prototyping are still the main approach used in industry. Such simulation- or prototype-based methods, which focus on sampling dynamic runtime information, are costly, and often difficult, to detect design flaws, due to their need of, and hopefully exhaustive, examination of all cases [1]. To be used at an early design stage, the efforts on simulation and prototyping have to be duplicated through every development stage when the design evolves with implementation details. Since they are not designed for collaborative applications, the analysis algorithms used at different stages usually require additional efforts to integrate, if such an integration is necessary.

As the development methodology for real-time embedded systems shifts from "capture-and-simulate" to "describe-and-synthesize" [2] combined with a model-drive architecture [3], model-based design and analysis methods have been studied extensively in both industry and research communities. Such a methodology shift provides a great opportunity for collaborative application of various analysis algorithms to design models and for systematic derivation of implementation details. Using model-based collaborative analyses in real-time ESCW development requires a framework with two key components: an expressive modeling language and a mechanism for interfacing analysis results. The modeling language plays the role of both information representation and data integration. The former determines the information needed for a target analysis and how to capture it, while the latter determines how to interface different analyses in the integration within the same design stage and across different design stages. The mechanisms for interfacing analyses allow the desired analysis to be plugged in a development tool chain and support the information flow between the various analyses.

Many existing general-purpose modeling languages, such as UML [4], that can be used to capture and represent needed information for analysis. However, more and more of today's analysis methods are implemented based on domain-specific modeling languages (DSML). A DSML restricts the information captured and exposed to a domain, reducing the complexity and improving the performance of the analysis algorithms. Many language frameworks have been developed and implemented to support the construction of a DSML, including OMG Meta Object Facility and UML Profiles [5], MetaGME in Generic Modeling Environment (GME) [6], and Eclipse Modeling Framework [7]. For real-time embedded software, standardized DSMLs also exist to support system analysis. Examples include UML Profiles by Object Management Group (OMG), Architecture Analysis and Design Language (AADL) by SAE [8], and EAST Architecture Description Language (EAST-ADL) initiated in Europe [9]. The metamodels and templates defined in AUTOSAR [10]—a popular standard for the automotive domain—is also a DSML. For timing and schedulability analyses, which are commonly required for real-time embedded software design and implementation, ad hoc DSMLs, such as SCADE/Lustre [11], ROOM [12], HRT-HOOD [13], AIF [14], and MetaH [15], have also been used. Unlike standard DSMLs that can be implemented in many modeling environments, these ad hoc languages are typically tied with their modeling environments and tools. While a standard DSML has better reusability and portability, an ad hoc DSML can be cleaner with fewer modeling elements and clear semantics, and is thus more effective and efficient.

The mechanisms for analysis interfaces include both the interfaces to invoke and execute an analysis algorithm and the transformation of input and output data to the required formats. Although it is desirable to have all analysis algorithms implemented in a common DSML with the same data format, such implementations are not always possible since i) the analysis algorithms are usually independent of their uses and the development process, and ii) different data with the corresponding semantics for these analysis algorithms are likely to make the common DSML too complex to be manageable and maintainable. Data-format transformations are, therefore, inevitable for integration of collaborative analyses. Depending on the interfacing mechanisms, the implementation of an analysis can be loosely-coupled or tightly-integrated. A loosely-coupled analysis can be integrated with a chosen modeling environment and other analyses using a framework, such as a client-server model, which allows easy and flexible integration of multiple analyses. An example framework supporting a loosely-coupled analysis is the Open Tool Integration Framework (OTIF) [16]. With OTIF, each analysis can be implemented and invoked as a standalone program using its own interfaces. The framework provides the translators for each analysis to transform the input data and analysis results to the required formats. The invocations and executions of an analysis are achieved through OTIF tool adapters implemented using CORBA middleware. Another example framework is the integration framework for open tool environment [17], whose implementation also uses CORBA with a configuration language to describe the interaction of analyses. A tightly-integrated analysis, on the other hand, has dedicated implementation to interact with other analysis and modeling environments. As a result, the analysis must be implemented with consideration of the interfaces and data formats used by the collaborative analyses and the host modeling environment to ensure the integrability. Although such an integration is fixed, it is usually easy to implement and deliver better performance for both individual analyses and their integration using such a framework than the one using a loosely-coupled one. Most timing and schedulability analyses used in current practice adopt a tightly-integrated approach for their integration with the modeling environment and other analysis algorithms. Examples include both commercial tools such as RaphidRMA [18] and Symta/S [19] and research tools such as TimeWeaver [20], Metropolis [21], VEST [22], and DESERT [23].

Many tool suites have been developed in recent years under various government-sponsored programs such as DARPA PCES, DARPA MoBIES, and NSF ITR, to integrate analysis in an end-to-end process for rapid real-time ECSW development. Honeywell created a toolset including ControlH and MetaH, where MetaH integrates system analysis and code generation [24]. MetaH uses its own domain-specific architecture description language and provides its own modeling environment with built-in analysis algorithms. VEST (Virginia Embedded System Toolkit) is a toolset developed for component-based software composition and analysis [22]. A domain-specific modeling language, VEST Perspective Aspect Language, is used to capture cross-cutting system properties, such as timing and security. VEST uses the Generic Modeling Environment (GME) [6] as its modeling environment and to define its modeling language, and the analysis algorithms, implemented as aspect checks, are integrated in the GME. Metropolis [21] provides an environment with a modeling language based on its own metamodel and a set of analysis algorithms for exploration of architectures. Time Weaver [20] and TimeWiz form another tool suite. Using this tool suite, the system models are captured in Time Weaver using the tool native modeling language. Built-in algorithms are created to construct dependency and response chains. Some properties such as timing and schedulability can be analyzed by TimeWiz that communicates with Time Weaver. The Automatic Control in Distributed Applications (AIDA) toolset integrates the design and analysis of embedded real-time control systems [25]. AIDA uses a domain-specific modeling environment to capture the system models. Its analyses are integrated in the modeling environment.

Besides the tool suites integrating modeling environment and analysis algorithms, there exist a large number of tool suites supporting analysis and verification based on simulation. Matlab [26] is a commercial product including tools for control design, simulation (e.g., Simulink/Stateflow), and code generation (e.g., Real-Time Workshop). All tools in Matlab share the Matlab modeling language with its graphic modeling environment. There also exist tools for simulation including distributed control software, such as TrueTime [27], implemented in Matlab with its graphic modeling environment and extending the modeling language via defining a library for computing platform components (hardware, operating systems, and well-known networks). Other simulation tools developed in the research community for system analysis and verification include Charon from University of Pennsylvania [28], Ptolemy from University of California at Berkeley [29], and BIP with THINK from VERIMAG [30]. All these tool suites have their own modeling languages and modeling environments, with the simulation performed by built-in backend algorithms.

The approach proposed and implemented in the AIRES toolkit is very different from these existing tools. Aiming at the integration of collaborative analyses, the AIRES toolkit implements a framework with mechanisms to integrate and interface various analysis algorithms used in the same development stage or across different stages. The key feature of this framework is that it allows the analysis algorithms, developed and implemented using their own language syntax, to collaborate semantically by sharing some common concepts, and thus facilitates engineering activities, such as design refinements and system verification. This enables the decoupling of collaborative analyses so that they may be implemented independently with their own strategies while relating to each other for later integration. Such a decoupled approach in AIRES provides a different, and unique, solution to the embedded real-time control software development requiring collaborative analyses. By contrast, other existing solutions use either i) a tightly-coupled approach that requires the analyses implemented based on a uniform model and tool, or ii) a loosely-coupled approach that allows the analyses implemented with different languages and different tools without explicit definitions of shared concepts and relies on the semantics-preserving model translation and availability of compatible tool interfaces, if any. As a result, the analysis algorithms using the AIRES framework, each with an independently-chosen implementation, can be integrated and used collaboratively without changing their implementations. A selected set of independently-implemented, collaborative analyses are integrated in the AIRES tool through meta modeling concepts to demonstrate this. Moreover, the analyses implemented in other tools are usually for post-design checks, whereas the AIRES toolkit supports the automatic design refinements using the analysis results. Such an analysis-based automatic design refinement is key to rapid ECSW development. Although the analyses presented in this paper are for system timing properties, the framework and principles are general and applicable to the integration of collaborative analyses for other system properties.

III. MODELING LANGUAGE AND SYSTEM MODELS

Most control functions in today's large real-time embedded systems run on a distributed computing platform, consisting of multiple computing devices and communication networks. In the ECSW development process, one of the key components is to derive a deployment model, also known as a runtime architecture model, with properly-chosen computing devices and scheduling parameters. In the derived deployment model, the software is typically organized as "tasks" and implemented as operating system threads or processes. The deployment decisions are critical for meeting the non-functional requirements, particularly timing constraints. For simplicity without losing generality, we use this step to demonstrate how AIRES supports the integration of collaborative analyses. The steps in the other stages of development may use the same framework with proper extension or replacement of the modeling languages and analyses to include other disciplines and system aspects. Given the fact that the development process must be defined with consideration of available analyses, our solution enables the integration of selected analyses that have already been implemented, instead of determining the needed analyses in a process followed by an implementation. Such a solution is thus more adaptive to large, evolving domains with multi-tier, multi-party development, where development methods and tools are commonly determined by the participating organizations.

To generate a deployment with model-based methods, we assume that the input of this design step is a software model that implements the designed control functions, and a computing platform model with processors, networks, and supporting software such as OS, middleware, protocol stack, etc. The output of this step is a deployment model. Selected analysis algorithms are required to collaborate in this step. To support the collaborative analyses, integrated and consistent data must be transferred among the analyses. AIRES uses a domain-specific modeling language that defines the essential information to perform the analyses to achieve this.

To illustrate the domain-specific modeling concept, we partition the ECSW designed at the deployment step into different domains, including software components, software structure, platform configuration, and runtime architecture. A modeling language, implemented as a metamodel, is defined for each domain.

A. Software Component Metamodel

Software components are basic building blocks in software domain, which are used to implement control functions. The software components may be reusable commercial-off-the-shelf (COTS) products or specially-developed in-house entities. The metamodel for software components in the AIRES is shown in Fig. 2.

According to the software component metamodel, every software component used in AIRES for analysis consists of an action, a set of ports, a process to execute the components. The action defines the behaviors that the component implements, which can be specified in other tools, such as Simulink/Stateflow in the form of mathematical equations and/or state machines. Such behavior specifications allow math-based or state-based analysis methods used for the software component verification. The software ports are used to specify the interactions between software components, and are classified into event and data ports for representing interactions with and without execution trigger, respectively. These ports are further divided into input and output ports. The CPU is used to specify the deployed process for the software component's execution. This allows the software component to be used in a design model where the deployment has not yet been determined, and to be assigned to only one CPU in a deployment model.

The AIRES software component metamodel also defines the attributes needed for the analyses under consideration, including resource demand, importance, and the required memory. The resource demand defines the computation resource consumed to execute the component, and can be represented in an abstract format before the deployment is determined. The abstract resource demand will then be represented in concrete values of execution time. The importance attribute is used to sequence independent components that co-reside in the same task. The required memory is used to check the memory constraint on a processor.

B. Software Structure Metamodel

A software structure model captures the dependencies and communications among the software components in realizing a control process. The ECSW for a control system may contain multiple control loops or control processes. In AIRES, each control process is represented as a transaction, which consists of interacting software components and forms an acyclic, direct graph. The whole ECSW can then be modeled as a set of concurrent transactions.

The performance metrics in the software structure model include both end-to-end response delays of the transactions and system-resource demands. The transactions with dependencies are transformed into a form that can be analyzed with existing timing analysis techniques The structural model is first transformed to a set of directed acyclic weighted graphs of transactions with single-input and single-output. The cycle elimination is achieved by separating the inner cycles for a transaction, relinking the feedback link to a dummy component, and assigning a new invocation rate for it. A transaction with multi-input and multi-output can be converted by creating a dummy component for start and a dummy component for end.

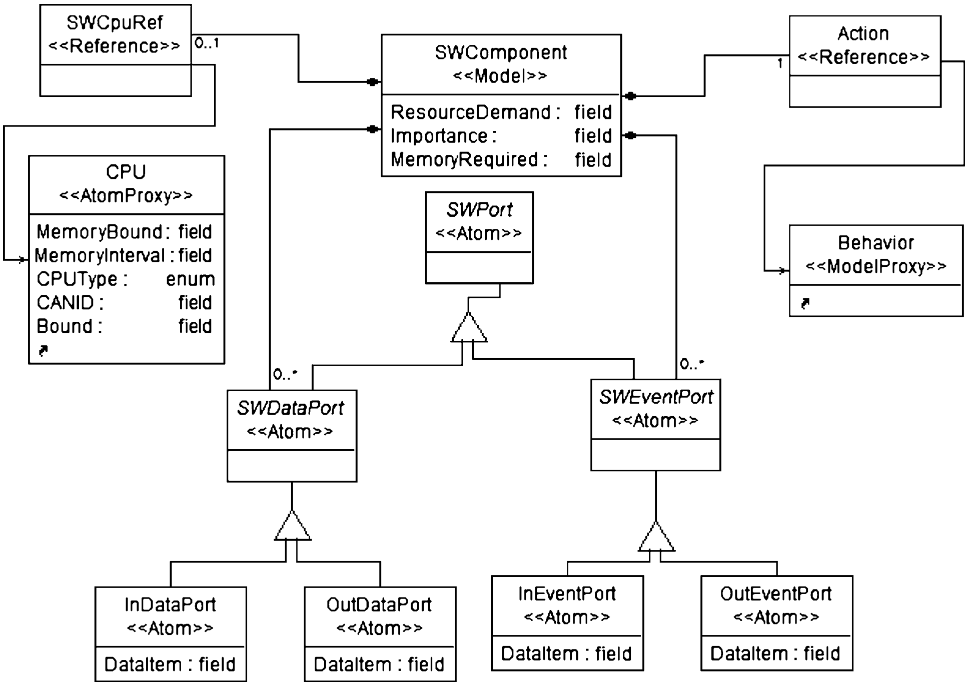


Fig. 2. ***AIRES software component metamodel.***

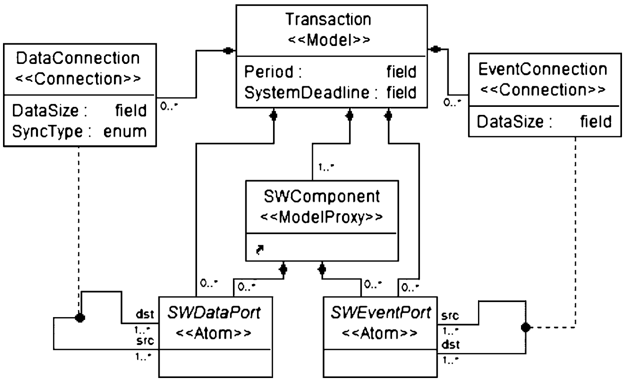


Fig. 3. ***AIRES software structure metamodel.***

The end-to-end response delay bound of a transaction and its system-resource demands are then decomposed into those on each node and link that the transaction runs. The annotations of performance parameters in the acyclic, direct graph at a higher-layer software structure model allow the performance analysis to evolve along with this hierarchical decomposition, using the refinement modeling method. As the model is refined, the constraints are partitioned and distributed over the lower-layer models, thus reducing the overhead of regenerating the performance modeling information. During the analysis, the derived constraints are used to compare with the performance characteristics of the model at a refined layer.

To support the analyses integrated in AIRES, we define a metamodel, shown in Fig. 3, to capture the software structure and the required attributes for analyses.

According to the metamodel, a transaction contains one or more software components that are connected through their ports. Depending on the types of ports used in the specification, the connection between two components can be either data- or event-based. While an event-based connection is synchronous, a data connection can be either synchronous or asynchronous. The ports in a connection must be of the same type. A transaction may be specified with ports, and interact with other transactions through their port connections. This allows to form a hierarchical system organization, which is simplified with only one level in this paper to reduce the complexity of traversing multiple levels in the analysis implementations. The information required in an analysis is also captured as a transaction's attributes, including its period and deadline. The modeling constructs of software component *SWComponent* and port *SWPort* link the software component metamodel and the software structure metamodel, allowing for the analyses across these domains, such as validating the behavior of a transaction.

C. Platform Configuration Metamodel

A computing platform specifies the computing and communication devices, as well as the supporting software, such as OSes, middleware and their services, device drivers, and network protocols, in a real-time embedded control system, which provides the resources for the execution and dynamic management of ECSW. It is a key subsystem as the thus-provided resources directly affect the performance and execution of ECSW. Depending on the resource availability and its usage strategies in a platform configuration, different system-level control performances may be achieved for the same set of control functions through different organizations of ECSW (e.g., running software components on different processors and/or executing them in different orders). Therefore, the platform configuration model is essential for analyses during the generation of the ECSW deployment model in order to meet the system requirements without exceeding the capacities of resources available in the platform. For this, AIRES defines a platform configuration metamodel, as shown in Fig. 4, which is designed to capture the platform configuration required by the analyses during the deployment model generation.

The metamodel defines modeling constructs for processors, network links, and OSes. For simplicity, we omit the constructs for middleware, sensors, and actuators. The cardinalities in the metamodel indicate that the platform must

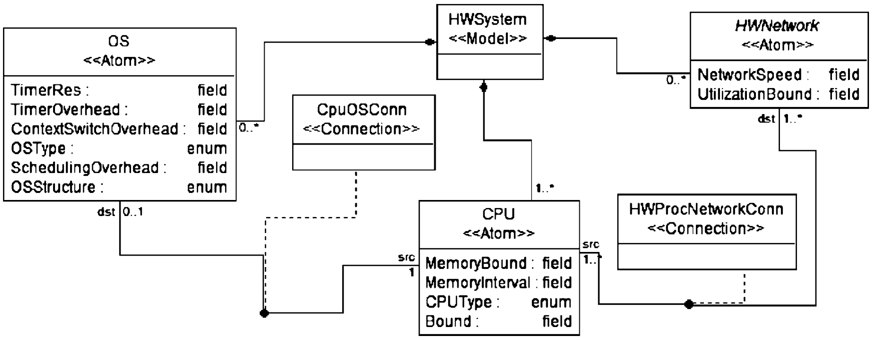


Fig. 4. ***AIRES platform configuration metamodel.***

contain at least one processor with at most one OS running on it. The metamodel is rather simplified compared to the platforms in practice, however, it contains the minimum requirements for executable run-time models as a system. The models of software architecture and of the platform are designed separately and integrated to generate the model of the system at the deployment design phase. The generation of deployment can be viewed as a refinement process and it requires only the software architecture with resource demands. This allows us to analyze the system such as obtaining the estimated performance that is useful for comparing software architecture designs and designing a platform even when the platform design is incomplete. A network may, or may not, be present in a platform. If a network exists, multiple processors can connect to the same network, and a processor is also allowed to connect to multiple networks (as a gateway, for example). All constructs for OSes, processors, and networks are defined with the attributes relevant to computing resources and timing, which are necessary for our analyses. Although the metamodel is simple, one can easily extend it to support modeling advanced platforms, such as the one with multi-core processors and/or with multiple OSes on each processor, or the analyses of other properties, such as power consumption. The processor construct *CPU* creates the linkage between the software component metamodel and the platform configuration metamodel, which allows the analysis, such as the one used to determine the processor for a software component execution.

D. Runtime Architecture Metamodel

A runtime architecture model captures all implementation details of ECSW on a given computing platform, including the deployment and execution parameters of all software components. Such an architecture model is essential for the analysis of system properties, such as timing and schedulability, which is a critical step in meeting the important timing and resource constraints in the final implemented system. Further, the model is usually used for model-based, automatic code generation. To capture the information needed for creating a runtime architecture model, AIRES defines a runtime architecture metamodel, as shown in Fig. 5.

The key modeling concept in the AIRES runtime architecture metamodel is the task. A task is the basic schedulable unit on a platform. In AIRES, a task can be implemented as a thread or process of an operating system. It consists of a sequence of software components. The value of the sequence number attribute of a software component indicates the order in which the software component should be executed in the task. When a task is activated, it executes its software components in their required order. It is possible to define a task as a placeholder before the runtime architecture is fully determined, which may contain no software component, as defined by the cardinality. A task must be assigned to one and only one processor for execution, and may contain input and output ports used to specify a chain of tasks. The AIRES runtime architecture metamodel defines a set of attributes to specify

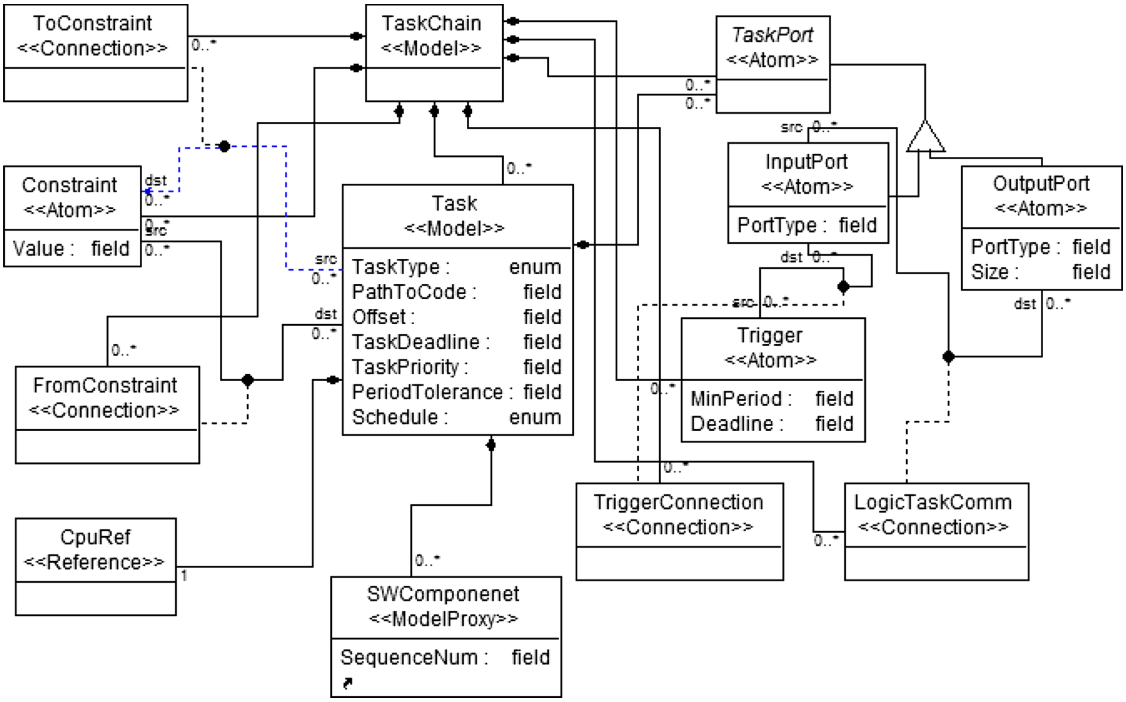


Fig. 5. ***AIRES runtime architecture model.***

the information needed to generate a deployment model, execute the analysis algorithms under consideration, and generate implementation code with the thus-determined invocation and execution parameters. Note that the metamodel requires the information that cannot be derived from other sources. However, the information essential for analyses can be derived from other models. Task execution times, for example, can be computed by adding the execution times of all contained software components. This enables us to avoid the possibility of duplicated and inconsistent specifications, and makes it easier to maintain the models.

The task chain is another important modeling concept defined in the AIRES runtime architecture metamodel. A task chain is specified as a set of tasks, perhaps distributed on different processors in the platform and communicating through their input and output ports. It represents an end-to-end information processing flow corresponding to a control process. A trigger, either a regular timing signal or an irregular event, arriving at the input port of a task chain activates the chain, and sequentially activates the execution from the first task to the last one. End-to-end timing constraints may also be specified for a task chain using the constraint modeling construct, along with the start and the end tasks indicated by *FromConstraint* and *ToConstraint*, respectively. The system-level end-to-end timing constraints are distributed over the software components to be used for constructing a feasible schedule. Then, the software components are merged into tasks while taking into consideration of both schedule flexibility and minimization of resource consumption, such as memory size, number of processors, network bandwidth, and so on.

In ARIES, the runtime architecture metamodel is related to the software component metamodel through the software component construct *SWComponent*, and related to the platform metamodel through the processor construct *CPU*. Such relations allow the analyses to form tasks and determine the scheduling and execution parameters.

Since metamodels are used as integration support for independently-developed analyses in AIRES, it is essential that the modeling environment chosen to implement AIRES is capable of defining a user-specified DSML. AIRES chooses the Generic Modeling Environment (GME) [6] developed at Vanderbilt University as its graphic modeling environment. GME is a configurable modeling tool supporting domain-specific modeling and synthesis. It can be configured with multiple metamodels, each of which defines a modeling paradigm (modeling language) of an application domain. The analysis and model-transformation algorithms can be implemented as loadable modules. GME is chosen based on the following requirements of the AIRES tool implementation: 1) modification of the modeling framework to contain the information needed for analyses; 2) integration of third-party algorithms provides hooks to allow the algorithms to access the models; and 3) visualization of analysis results to visually verify the results generated by the analysis algorithm.

IV. INTEGRATION OF ANALYSES IN AIRES

The AIRES framework takes a decoupled—instead of tightly- or loosely-coupled—approach as its strategy to support integration of collaborative analyses. The decoupled approach allows the analysis algorithms to use their own modeling concepts but requires accessible interfaces in their implementations. The collaboration and integration of these analysis algorithms are then achieved through the metamodel associations, which can be defined after implementation of the analysis algorithms, thus allowing for independent development and implementation of analysis algorithms, while still achieving collaborative analyses. Using the decoupled approach for integration requires a powerful modeling environment that provides the capability of implementing user-defined DSMLs.

A difficulty associated with AIRES in integrating collaborative analyses is the circular dependencies of information among the analyses. For example, the resource demand of a software component, represented in the form of worst-case execution time (WCET) and required by the analysis in the runtime model generation can only be determined after knowing the processor to execute the component. This circular dependency is addressed by using analysis-based, iterative model refinements in AIRES. With different types and configurations of the underlying system services, the performance of the runtime model can be dramatically different. The model refinement is important when some performance constraints are violated. This requires the performance analysis of runtime model. Our runtime model performance analysis is based on the timing and schedulability analysis for real-time systems, which are specific to the scheduling algorithms. In particular, the assignments of scheduler configuration parameters, such as scheduling policy, task invocation mechanism, and the priority for each task, are iteratively refined to yield a feasible schedule of the system. Task timing constraints and device resource constraints are verified based on the analysis of the task set on each device. The analysis reveals the resource consumption on a single device and link, and the responsiveness of individual tasks. The analysis results are used for further model refinements.

The implementations of AIRES analyses are tied to the modeling environment, which is similar to the tools using the tightly-coupled approach. However, an analysis in AIRES is implemented based only on its own defined metamodel and with interfaces interacting with its host modeling environment. The interfaces allow the analyses to directly access the model data in the modeling environment, thus improving performance over the loosely-coupled approach. Such interfaces can be implemented by adding wrappers to the existing analyses. GME provides three interfaces to integrate customizable algorithms: add-ons, plug-ins, and interpreters. The add-ons and plug-ins are based on the OCL (Object Constraint Language), and are suitable for simple constraint checks. Our interface implementations use the interpreter mechanism that can perform complex operations. The

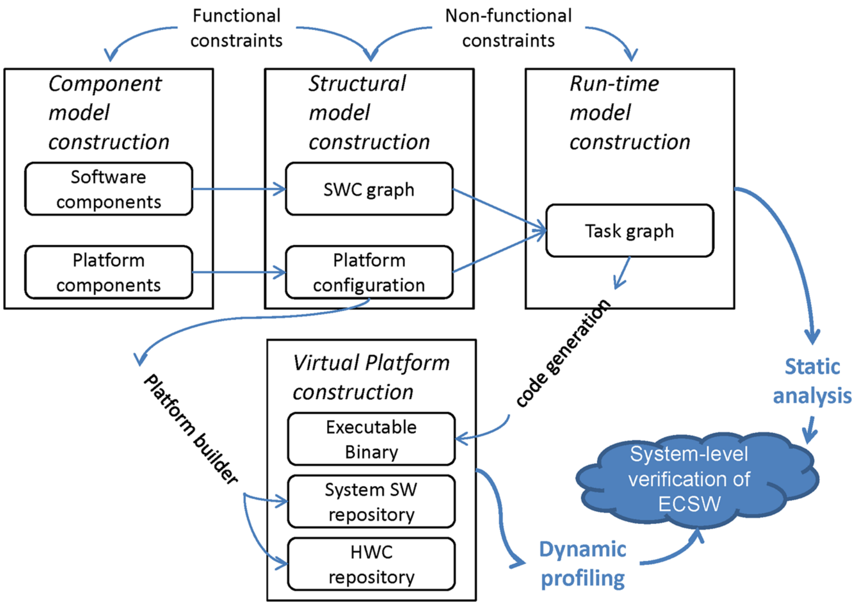


Fig. 6. *The integration framework for system-level verification of ECSW.*

interpreters are built as dynamic link libraries. Specifically, our interpreters are built with the GME Builder Object Network (BON) interfaces. Upon its invocation, the interpreter creates a data structure that contains a mirror object for every modeling element in the model. Other operations in the interpreter can then access the full model through the data structure and its methods.

AIRES implements a set of algorithms with the defined metamodels and the decoupled analyses to support automatic, rapid ECSW development. These analyses focus on timing and resource guarantees, and include the algorithms for runtime architecture model generation, the algorithms for timing and schedulability analyses, and the algorithm for code generation and profiling on a virtual platform. Fig. 6 shows the overall process with modeling data and analysis algorithm implemented in AIRES.

A. Analysis Integration for the Runtime Architecture Model Generation

The runtime architecture model generation creates a runtime architecture model using a software structure model and a platform configuration model. All the models are defined using the AIRES domain metamodels. The generation must identify which processor or network link in the platform model to execute which software component or transfer which data in the software structure model, group the software components to form tasks for execution, and assign tasks' properties for an operating system to schedule them at runtime. Analyses are required during such a generation to ensure the workload on each processor or communication link within its capacity, and the system-level timing constraints can be met with the generated runtime architecture. Given all design artifacts are captured in models, the generation implemented in AIRES can be considered as an analysis-guided model transformation, which is overviewed in [14], [31].

The generation is implemented as a sequence of model transformation steps, as illustrated by the example in Fig. 7. The analysis algorithms used in the generation include the branch-and-bound (B&B) with forward checking [32] for component allocation, rate similarity with component sequencing for task formation [31], [33], and

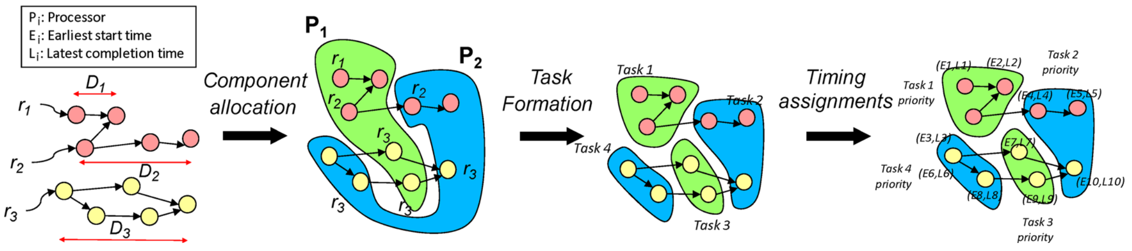


Fig. 7. *The runtime architecture model generation.*

timing & priority assignments for dependent tasks using simulated annealing with the latest task completion time [34]. The B&B with forward-checking algorithm assigns the software components to a process one at a time, followed by the analysis of the resource utilizations and estimation of an optimal assignment for the remaining the components. The algorithm is implemented based on its metamodel mapped to the AIRES software structure metamodel, with the processor information from the platform configuration model. The results are reflected as the values assigned for the *CPU* reference of each *SWComponent* in the software structure model. In addition, some classic analysis-based allocation methods, including first-fit, load-balance, communication minimizing with k-way min-cut, have been implemented in the same way in AIRES to meet different needs in various designs. Similarly, the task formation uses the analysis of the invocation rates captured in the software structure model, and group them with an analysis that minimizes the potential overheads introduced by indirectly dependent components. The analyses are implemented based on their metamodels mapped to the software structure metamodel and the runtime architecture metamodel. The results are reflected as the values assigned for the *SWComponent* and their *SequenceNum* of the *Task*. The final timing and priority assignments are determined by the timing-assignment step, which uses analyses with iterative priority assignment in [34]. The analyses are implemented based on their metamodels mapped to the runtime architecture metamodel, and the results are reflected as the values assigned to the attributes of *Task*. All these analyses are implemented to interface with the GME modeling environment directly to access and modify the model data, and are invoked one after another automatically during the runtime architecture model generation using the invocation methods provided by the GME.

B. Schedulability Analysis for Design Refinement

Schedulability analysis is essential to the verification of end-to-end timing constraints and resource usages after the generation of a runtime architecture model. Although these constraints are considered during the generation, the decisions, such as priority assignments and task formation, are made for each individual processor. The design can also be improved with refinements, such as dependency elimination to achieve better scalability and performance.

The schedulablity analysis integrated in AIRES is based on a classical worst-case response time analysis [35]. It uses the worst-case task execution times, identifies the worst-case instant when a task starts, and creates a busy period to compute the response time for each task. With the AIRES runtime architecture model capturing distributed ECSW, the algorithm has been extended to a holistic end-to-end analysis, including the messages passed through the network. The computed end-to-end response time for each task chain is compared with its timing constraints, and the design passes the analysis if the response time of every task chain is less than the chain's constraint. The implementation of schedulability analysis is based on a metamodel mapping to the AIRES runtime architecture metamodel, and is integrated in the GME modeling environment using the GME interface for invoking external functions. Although the schedulability analysis implemented in AIRES is based on static worst-case execution times of software components, the AIRES framework allows a more powerful analysis algorithm that computes the execution times resulting from the dynamic behaviors to be integrated, as such an algorithm can trace the software behavior from the task to the software components to its own behavior.

Scheduling the task chains with dependencies in a distributed environment is difficult because of the complexity introduced by the dependencies. Such task dependencies in a runtime architecture model come from the transactions in a software structure model, which in turn come from control loops. Eliminating task dependencies, while keeping the data consistency, can improve the flexibility of system configuration to include new software when the system evolves, and simplify the design and scheduling by applying a classical analysis for independent tasks. AIRES introduces a shared buffer approach, along with the method for splitting dependent tasks into buffer polling and data computation segment with their new invocation frequencies, to eliminate the dependencies while preserving data consistency and timing constraints. Analyses are used to determine parameters, such as buffer size and polling task frequency. Figs. 8 and 9 show how a set of dependent tasks are transformed to an independent set of tasks polling the shared buffers at predefined intervals. The task graph of the original system contains 4 tasks, where T_3 depends on the outputs of both T_1 and T_2, and T_4 depends on the output from T_3 as in Fig. 8. The system can be transformed into a system with independent tasks with the shared buffer approach. The transformed system contains the original 4 tasks and additional 3 shared buffers. The polling tasks with invocation rates, r_3 and r_4 are assigned to T_3 and T_4, respectively, such that the rates are faster enough for successor tasks preserve the correct data while the timing constraints, D_1 is satisfied. (see [36] for more details). The methods and analyses thereof are implemented independently and then integrated using the

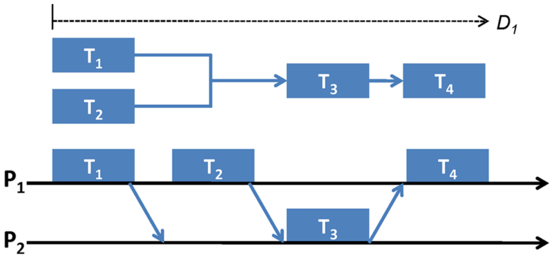


Fig. 8. *An example set of dependent tasks.*

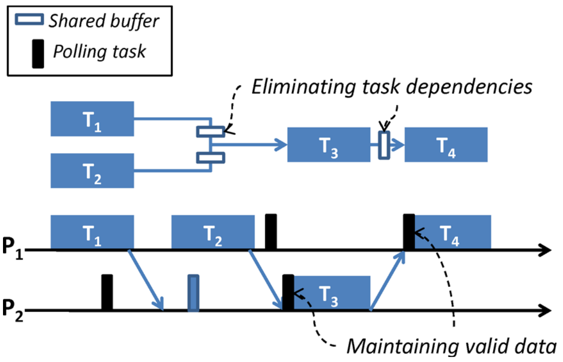


Fig. 9. ***Tasks with shared buffers after dependencies are eliminated.***

AIRES framework. The integration is done by mapping the methods' internal data structures to the AIRES runtime architecture model and the interfaces of GME for external functions.

C. Integration of Profiling Using a Virtual Platform

The analyses used in AIRES are traditional worst-case analyses. While worst-case analyses can provide absolute guarantees, they usually yield pessimistic results and low utilization of resources. However, the analysis does not provide the information at run-time, such as resource utilization or response time for a specific period that may be useful for the fine-tuning of a system. An efficient development of a system, however, still requires performance profiling of a system at run-time. To avoid the pessimistic results, it is critical to capture the non-functional characteristics, such as timing and resource usage. Since it is extremely difficult to do this mathematically, the simulation-based profiling is required for such capturing dynamic behaviors of a system and AIRES integrates a simulation-based profiling on a virtual platform to obtain realistic values for the ECSW execution. To our knowledge, there is no previous or ongoing research on systematic platform performance modeling for system analysis and platform assessment.

AIRES uses functionality-correct and timing-accurate software to realize the hardware components of processors, memory subsystems, buses, etc. The software that performs the functionality of a complete hardware configuration is considered as a virtual platform and can be used for accurate simulations. With the AIRES framework, a virtual platform is constructed with existing components in the hardware component repository according to the platform configuration model.

Similarly, the supporting software, such as operating systems and network protocols, can also be configured using the software components in the repository and the board support packages (BSPs) for different processors.

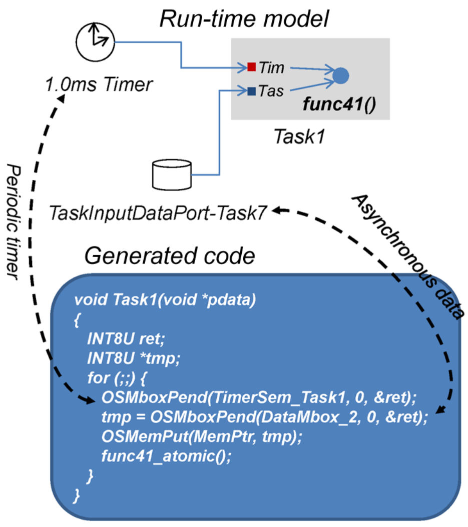
The virtual platform corresponding to the platform configuration model can then be simulated using the VaST system—a cycle-accurate, highly-configurable simulation tool [37]. The VaST system runs as a separate tool, communicating with AIRES through the platform configuration model. To perform simulation on a virtual platform so that the runtime architecture model can be profiled, AIRES integrates a code generator to automatically create executable code of the designed runtime architecture model on the specified platform configuration. The method used in the generator ensures that only necessary code is generated. Fig. 10 shows an example of code generation from a given runtime architecture model.

The generated code with its virtual platform can then be simulated on the VaST system. To measure the runtime information of interest, such as execution times and scheduling overheads, AIRES uses a separate tool for system monitoring. The system monitoring tool stores traces and processes the data. It communicates only with the virtual platform, but not directly with AIRES. Details on this virtual platform and its integration with AIRES can be found in [38].

V. CASE STUDY: ELECTRICAL THROTTLE CONTROL SOFTWARE DEVELOPMENT

To illustrate how the techniques implemented in the AIRES toolkit support rapid ECSW development, we present a case study of using AIRES for electrical throttle control (ETC) software development. To simplify the discussion, the real ETC has been sanitized with only the components, properties, and their interactions sufficient to show the modeling framework and the integrated collaborative analyses.

The development of the ETC starts from the control design, which focuses on capturing the functional behaviors. The ECSW development starts after the completion of the control design, and transforms the designed controls into a set of transactions, each of which realizes some control functions and consists of software components.



```
void Task1(void *pdata)
{
  INT8U ret;
  INT8U *tmp;
  for (;;) {
    OSMboxPend(TimerSem_Task1, 0, &ret);
    tmp = OSMboxPend(DataMbox_2, 0, &ret);
    OSMemPut(MemPtr, tmp);
    func41_atomic();
  }
}
```

Fig. 10. ***An example of automatic code generation.***

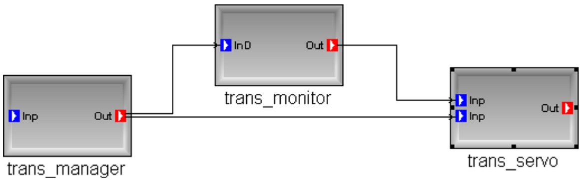


Fig. 11. ***Transactions in the ETC.***

Table 2 The Definitions of Transactions

transaction	period (ms)	components
trans_manager	10	holder, state_detector
trans_monitor	40	input_proc, tps_proc, actuator_proc, sybsys_proc, merge
trans_servo	3	control source_proc, driving, cruise, min, post_control, cmd_gen

Fig. 11 shows the transactions and their interactions in the ETC software, with the components listed in Table 2. Fig. 12 shows the monitor transaction model *trans_monitor* in AIRES, which contains the event port connections between *input_proc* and *actuator_proc* and between *actuator_proc* and *merg*. All other connections are made through data ports, and a transaction communicates with other transactions through the input data port *InMagsig* and the output data port *OutStatusData*. The behavior in each software component is linked to a control block in a different tool. As one can be see, the software model is specified using the AIRES software structure metamodel.

The platform executing the ETC is assumed to have two Electronic Control Units (ECUs) connected by a communication bus, as shown in Fig. 13. The platform model is captured using the AIRES platform configuration metamodel.

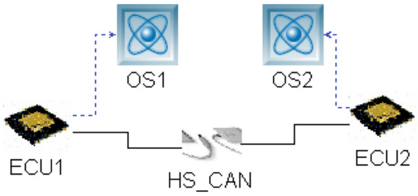


Fig. 13. ***The platform configuration model for the ETC.***

Table 3 The Generated Runtime Architecture Results

component	seq #	task‘	ECU
holder	1	Task_manager_ECU1	ECU1
state_detector	2	Task_manager_ECU1	ECU1
actuator_proc	3	Task_monitor_ECU1	ECU1
tps_proc	2	Task_monitor_ECU1	ECU1
input_proc	1	Task_monitor_ECU1	ECU1
subsys_proc	4	Task_monitor_ECU1	ECU1
merge	5	Task_monitor_ECU1	ECU1
source_proc	1	Task_servo_ECU1	ECU1
min	3	Task_servo_ECU1	ECU1
driving	2	Task_servo_ECU1	ECU1
control	1	Task_servo_ECU2	ECU2
post_control	3	Task_servo_ECU2	ECU2
cmd_gen	4	Task_servo_ECU2	ECU2
cruise	2	Task_servo_ECU2	ECU2

So far, only the platform-independent software model of the ETC and the platform configuration model are captured. The runtime architecture generation needs to be performed to assign the software components in the transaction onto the ECUs in the platform and to form the tasks or OS threads. This is achieved by a plug-in program that integrates the required, individually-implemented analysis algorithms. Table 3 shows the generated result from the application of the analysis of a communication-minimization allocation policy. The *seq* # column indicates the execution sequence of components in a task.

The resultant runtime architecture model is shown in Fig. 14, with one of the tasks *Task_servo_ECU1* shown in

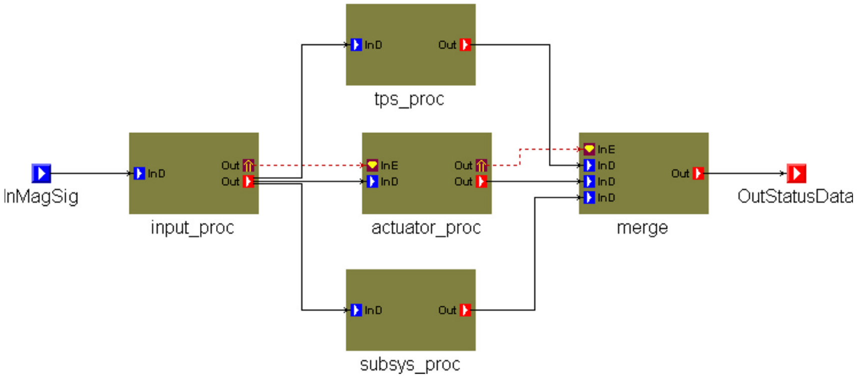


Fig. 12. ***The trans_monitor model.***

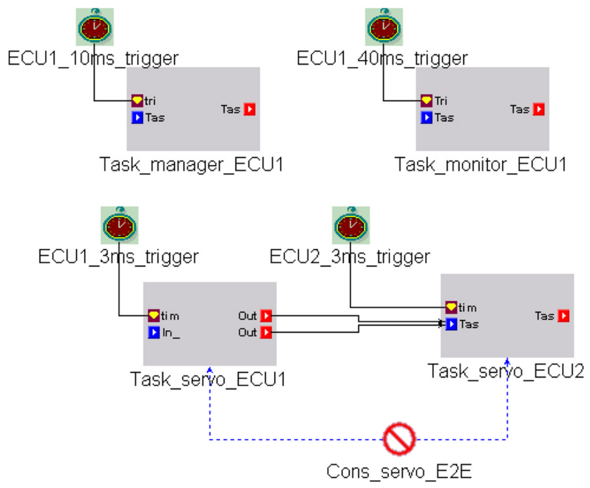


Fig. 14. ***The generated ETC runtime architecture model.***

Fig. 15. The components in a task are sequenced, and the ECU running the task is indicated. The thus-generated tasks are also triggered with the timing signals each of which is automatically assigned a trigger port that connects to a timing source. The resultant runtime architecture model is specified using the AIRES runtime architecture metamodel.

With the runtime architecture mdoel generated, one can perform analyses, such as timing and schedulability analyses, task refinement with shared buffers, and task profiling on a virtual platform. The AIRES toolkit implements common analyses, including rate-monotonic scheduling, deadline-monotonic scheduling, and manually-assigned priority scheduling. For dependent tasks in a task chain, we have also implemented algorithms, such as deadline distribution and end-to-end timing analysis. Table 4 shows the timing and schedulability analyses results under the rate-monotonic scheduling policy. In this table, *prio* represents task priority, *P* task period, *D* deadline, *wcet* the worst-case execution time, *wcrt* the worst-case response time, and *U* the workload introduced to the ECU utilization. The time is represented in milli-seconds, and a higher priority is represented by a larger number.

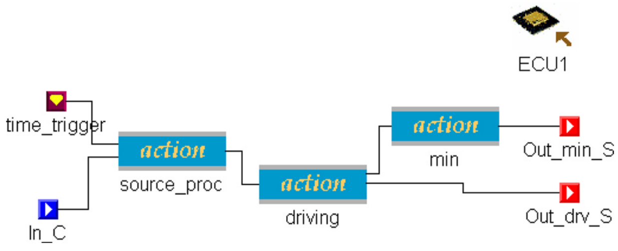


Fig. 15. ***The generated Task_servo_ECU1 task.***

Table 4 Schedulability Analysis Results

Task	prio	P	D	wcet	wcrt	U
task_servo_ECU1	5	3	3	0.6	0.6	0.2
task_servo_ECU2	4	3	3	0.8	0.8	0.27
task_manager_ECU1	2	10	10	0.53	1.13	0.053
task_monitor_ECU1	1	40	40	1.3	2.43	

Although *task_servo_ECU2* depends on *task_servo_ECU1*, the analysis shows that the worst-case response time of *task_servo_ECU2* is 0.8, implying that the task is invoked independently of *task_servo_ECU1*. To account for the dependency, we ran the deadline distribution, followed by the end-to-end timing analysis, which yields the deadline for *task_servo_ECU1* to be 1.28, and the offset of *task_servo_ECU2* to be 1.4. As a result, the new worst-case response time for *task_servo_ECU2* becomes 2.2. In case the system is not schedulable, the AIRES toolkit provides options to automatically refine the design by adjusting either task priorities, or task periods, or separation of dependent tasks [33], [34], [36].

As this case study shows, the AIRES toolkit allows integration of collaborative analyses at different design stages and supports design automation. The model-based approach and analyses verify the generated results at every step before proceeding to the next step, ensuring the correct design generated at every stage, thus supporting rapid development of the ETC and avoiding unnecessary interactions between later and earlier design stages.

VI. CONCLUSION

Rapid development of correct ECSW for large embedded systems, such as avionics or automotive controls, is becoming very challenging as the software for such systems becomes increasingly complex and requires multi-discipline and multi-group collaboration. Model-based methodologies, which provide high-level abstractions and allow system-level analyses, have been widely used to address this challenge. For a model-based methodology, automation and collaborative analyses are key to rapid and correct ECSW development. As traditional analysis methods are designed and implemented independently of the ECSW development process with their own modeling concepts, the main challenge in applying these techniques is the integration of various analyses in a model-based development process such that these analyses collaborate to make design choices in the automation. Moreover, many analysis methods require accurate knowledge of a complete design, making it difficult to apply them at an early design stage where the design automation starts. We meet this challenge by developing a well-defined framework with domain-specific modeling and analysis interfacing mechanisms. Specifically, we present such a framework that uses the domain-specific modeling language for an integrated data model to support the collaboration between different analysis methods. With such a modeling language, analysis methods are implemented using a decoupled approach with the interfaces to manage their invocations and executions. We have implemented such a framework in the AIRES toolkit to demonstrate its ability to support rapid and correct ECSW development via analysis-based design automation. To implement the AIRES framework, we chose the Generic Modeling Environment (GME) tool as the modeling environment because it supports specification of user-defined DSMLs and plug-in of individually-implemented analysis algorithms, which are essential for the implementation of the AIRES framework. The analysis methods integrated in the AIRES toolkit include those for software component allocation, task formation, and timing and priority assignments used in the runtime architecture model generation, those for schedulability and timing analyses used in design refinement, and the profiling methods based on a virtual platform for measuring dynamic runtime software. All of these analysis methods, implemented and integrated, work collaboratively via the data captured by the corresponding metamodels to support design automation in the AIRES toolkit. With the rigorous analyses applied to each step of the process and the automation, the AIRES toolkit is shown to facilitate rapid and correct ECSW development. ■

REFERENCES

[1] W. W. Peng and D. R. Wallace, *Software Error Analysis*. Summit, NJ: Silicon Press, 1999.

[2] D. D. Gajski, F. Vahid, and S. N. J. Gong, *Specification and Design of Embedded Systems*. Upper Saddle River, NJ: Prentice Hall, 1994.

[3] Object Management Group, *Mda Guide, Version 1.0.1,* 2003.

[4] Object Management Group, *Unified Modeling Language, Version 2.1.1.,* 2007. [Online]. Available: http://www.omg.org/technology/documents/spec_summary.htm

[5] Object Management Group, *Meta Object Facility Core, Version 2.0,* 2006. [Online]. Available: http://www.omg.org/technology/documents/spec_summary.htm

[6] Institute for Software Integrated Systems, *The Generic Modeling Environment.* [Online]. Available: http://www.isis.vanderbilt.edu/Projects/gme

[7] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose, *Eclipse Modeling Framework*. Boston, MA: Addison-Wesley Professional, 2003.

[8] Society of Automotive Engineers, *Architecture Analysis and Design Language (AADL), as5506,* 2006. [Online]. Available: http://www.sae.org/technical/standards/AS5506/1

[9] Information Technology For European Advancement, *EAST Embedded Electronic Arhictecture, Version 1.02,* 2004. [Online]. Available: http://www.sae.org/technical/standards/AS5506/1

[10] S. Furst, "AUTOSAR—An open standardized software architecture for the automotive industry," in *Proc. 1st AUTOSAR Open Conf. 8th AUTOSAR Premium Member Conf.*, Oct. 23, 2008.

[11] B. Dion, T. Le Sergent, B. Martin, and H. Griebel, "Model-based development for time-triggered architectures," in *Proc. 23rd DASC*, Salt Lake City, UT, Oct. 24–28, 2004, vol. 2, pp. 6.D.3–6.1-7.

[12] B. Selic, G. Gullekson, and P. T. Ward, *Real-Time Object-Oriented Modeling.* Hoboken, NJ: John Wiley & Sons, 1994.

[13] P. Cornwell and A. Wellings, "Transaction specification for object-oriented real-timesystems in HRT-HOOD," vol. LNC 1031, *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 1996, pp. 365–378.

[14] Z. Gu, S. Wang, and S. K. K. G. Shin, "Multi-view modeling and analysis of embedded real-time software withmeta-modeling and model transformation," in *Proc. 8th IEEE Int. Symp. HASE*, Tampa, FL, Mar. 2004, pp. 32–41.

[15] S. Vestal, *Mateh User's Manual*, 1998. [Online]. Available: http://www.htc.honeywell.com/metah/uguide.pdf

[16] ESCHER Institute, *Open Tool Integration Framework,* 2004. [Online]. Available: http://www.escherinstitute.org/Plone/frameworks/otif

[17] G. Paul, K. Sattler, and M. Endig, *An Integration Framework for Open Tool Environments,* 1996. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.8384

[18] Tri-Pacific Software, *Rapidrma: The Art of Modeling Real-Time Systems,* 2007. [Online]. Available: http://www.tripac.com/html/prod-fact-rrm.html

[19] A. Hamann, M. Jersak, K. Richter, and R. Ernst, "A framework for modular analysis and exploration of heterogeneousembedded systems," *J. Real-Time Syst.*, vol. 33, no. 1–3, pp. 107–137, Jul. 2006.

[20] D. de Niz and R. Rajkumar, "Time weaver: A software-through-models frameowrk for embedded real-time systems," in *Proc. ACM SIGPLAN Conf. Language, Compiler Tool Embedded Syst.*, San Diego, CA, 2003, pp. 133–143.

[21] F. Balarin, Y. Watanabe, H. Hsieh, L. L. C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An integrated electronic system design environment," *Computer*, vol. 36, no. 4, pp. 45–52, Apr. 2003.

[22] J. A. Stankovic, R. Zhu, R. Poornalingam, C. L. Z. Yu, M. Humphrey, and B. Ellis, "VEST: An aspect-based composition tool for real-time systems," in *Proc. 9th IEEE Real-Time Embedded Technol. Applications Symp.*, Washington, DC, May 2003, pp. 58–69.

[23] S. Neema, J. Sztipanovits, and G. Karsai, "Constraint-based design space exploration and model synthesis," in *Proc. 3rd ACM Int. Conf. Embedded Softw.*, Philadelphia, PA, Oct. 2003, pp. 290–305.

[24] B. Lewis and D. J. McConnell, "Reengineering real-time embedded software onto a parallel processing platform," in *Proc. 3rd IEEE Working Conf. Reverse Eng.*, Monterey, CA, Nov. 1996, pp. 11–19.

[25] O. Redell, "Response Time Analysis for Implementation of Distributed Control Systems," Ph.D. dissertation, Department of Machine Design, Royal Institute of Technology, KTH, Stockholm, Sweden, Apr. 2003.

[26] The MathWorks, *Documentation for Mathworks Products,* 2008. [Online]. Available: http://www.mathworks.com/access/helpdesk/help/helpdesk.html

[27] M. Ohlin, D. Henriksson, and A. Cervin, "Truetime 1.5—Reference Manual," Department of Automatic Control, Lund University, Sweden, Tech. Rep., 2007.

[28] Y. Hur and I. Lee, "Distributed simulation of multi-agent hybrid systems," in *Proc. 5th IEEE Int. ISORC*, Apr. 29–May 1, 2002, pp. 358–364.

[29] E. A. Lee, "Overview of the Ptolemy Project," University of California, Berkeley, Berkeley, Ca., Tech. Rep. Technical Memorandum No. UCB/ERL M03/25, Jul. 2003.

[30] M. Poulhies, J. Pulou, C. Rippert, and J. Safikis, "A methodology and supporting tools for the development of component-based embedded systems," in *Proc. 13th Monterey Workshop*, Paris, France, Oct. 2006, pp. 75–96.

[31] S. Wang and K. G. Shin, "Task construction for model-based design of embedded control software," *IEEE Trans. Softw. Eng.*, vol. 32, no. 4, pp. 254–264, Apr. 2006.

[32] S. Wang, J. R. Merrick, and K. Shin, "Component allocation with multiple resource constraints for large embedded real-time systems," in *Proc. 10th IEEE RTAS*, Toronto, Canada, May 2004, pp. 219–226.

[33] S. Kodas, S. Wang, and K. G. Shin, "Transforming structural model to runtime model of embedded softwarewith real-time constraints," in *Proc. IEEE DATE*, Munich, Germany, Mar. 2003, pp. 170–175.

[34] J. R. Merrick, S. Wang, K. Shin, J. Song, and W. Milam, "Priority refinement for dependent tasks in large embedded real-time software," in *Proc. 11th IEEE RTAS*, San Francisco, CA, Mar. 2005, pp. 365–374.

[35] J. W. S. Liu, *Real-Time Systems*. Upper Saddle River, NJ: Prentice Hall, 2000.

[36] S. Kodas, S. Wang, and Z. G. K. G. Shin, "Improving scalability of task allocation and scheduling in large distributed real-time systemsusing shared buffers," in *Proc. 9th IEEE RTAS*, Washington, DC, May 2003, pp. 181–188.

[37] VaST Systems, *VaST Tools and Models for Embedded System Design.* [Online]. Available: http://www.vastsystems.com/

[38] S. Park, W. Olds, K. G. Shin, and S. Wang, "Integrating virtual execution platform for accurate analysis in distributed real-time control system development," in *Proc. 28th IEEE RTSS*, 2007, pp. 61–72.

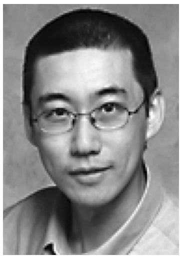
ABOUT THE AUTHORS

Sangsoo Park (Member, IEEE) received the B.S. degree from Korea Advanced Institute of Science and Technology, Daejeon, Korea in 1998 and received the M.S. and Ph.D. degrees from Seoul National University, Seoul, Korea in 2000 and 2006 respectively. He is now a full-time lecturer in the department of Computer Science and Engineering at Ewha Womans University, Seoul, Korea.

Kang Geun Shin (Fellow, IEEE) is the Kevin and Nancy O'Connor Professor of Computer Science and Founding Director of the Real-Time Computing Laboratory in the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan.

His current research focuses on QoS-sensitive networking and computing as well as on embedded real-time OS, middleware and applications, all with emphasis on timeliness and dependability. He has supervised the completion of 65 Ph.D. theses, and authored/coauthored about 700 technical papers (more than 250 of which are in archival journals) and numerous book chapters in the areas of distributed real-time computing and control, computer networking, fault-tolerant computing, and intelligent manufacturing. He has co-authored (jointly with C. M. Krishna) a textbook *Real-Time Systems* (McGraw Hill, 1997).

He has received a number of best paper awards, including the IEEE Communications Society William R. Bennett Prize Paper Award in 2003, the Best Paper Award from the IWQoS'03 in 2003, and an Outstanding IEEE Transactions on Automatic Control Paper Award in 1987. He has also coauthored papers with his students which received the Best Student Paper Awards from the 1996 IEEE Real-Time Technology and Application Symposium, and the 2000 UNSENIX Technical Conference. He has also received several institutional awards, including the Research Excellence Award in 1989, Outstanding Achievement Award in 1999, Service Excellence Award in 2000, Distinguished Faculty Achievement Award in 2001, and Stephen Attwood Award in 2004 from The University of Michigan; a Distinguished Alumni Award of the College of Engineering, Seoul National University in 2002; 2003 IEEE RTC Technical Achievement Award; and 2006 Ho-Am Prize in Engineering.



Shige Wang (Member, IEEE) received the B.S. and M.S. degrees from the Northeastern University, Shenyang, P.R. China in 1989 and 1995, respectively, and received the Ph.D. degree from the University of Michigan, Ann Arbor, in 2004. He is now a senior research scientist at General Motors R&D.

AUTHOR QUERY

No query