Component-based Approaches for Design and Analysis of Real-time embedded systems

A research report

Jagadish Suryadevara,
Ph.D student, MDH, Västerås, SWEDEN

ABSTRACT:

This report presents an overview of the field of research i.e. component based design and analysis of real-time embedded systems. A research approach for component based modeling and analysis of real-time embedded systems is proposed. The related approaches e.g. BIP framework, PTIDES, COMDES, etc are described and compared. Thus the report presents related research activities, including current research issues, state of the art and related work, as well as key conferences and leading research groups in relation to the proposed ph.d work.

INTRODUCTION

The increased complexity of embedded real-time systems, in terms of non-functional aspects such as timing and resource usage constraints leads to increasing demands with respect to analyzable high-level design such as architectural specifications [1,2,5]. This calls for methods, models, and tools that permit a controlled and structured working procedure during the complete life cycle of the system. The component based software engineering (CBSE), in spite of involved difficulties, seems capable to meet the challenges of development of predictable RT systems. A CBSE framework for predictable RT systems involves a specialized or domain-specific component model and associated tools for design and development of components, component repository, as well as various tools for formal analysis of non-functional aspects of individual components as well as the final composed system (e.g. schedulability). This report presents an overview of development of embedded systems in terms of characteristics of RT systems, existing methods, as well as challenges that needs to be met.

Real-time systems are usually used to control or interact with a physical system, and the timing constraints are imposed by the environment [2,5]. Real-time systems can be constructed of sequential programs, but typically they are built of concurrent programs, called tasks. Tasks are usually divided into periodic and non-periodic tasks. When a processor is to execute a set of concurrent tasks, the operation of the CPU must be assigned to the various tasks according to a predefined criterion called a scheduling policy. Various algorithms are currently available for scheduling of real-time systems. They fall into two categories: off-line and on-line scheduling. In off-line scheduling, the scheduler has complete knowledge of the task set and its constraints. Scheduling decisions are based on fixed parameters that are assigned to tasks (e.g. deadline, priority, WCET, ..etc)
before their activation. The off-line-generated schedule is stored and dispatched subsequently during run time of the system. On the other hand, on-line scheduling algorithms make their scheduling decisions during run time. Based on requirements for meeting timing constraints such as ‘deadline’, the systems are categorized as ‘hard’ or ‘soft’ RT systems.

Designing reusable real-time components is more complex than designing non-reusable, non-real-time components [1]. This complexity arises from several aspects of real time not relevant in non-real-time systems. In real-time applications, components must collaborate to meet timing constraints, also referred to as end-to-end transaction deadlines [5]. Timing analysis is performed at two levels, the task level and the system level. At the task level the worst-case execution time (WCET) for each task is analyzed or estimated. The analysis can be performed in two ways, either via measurement or by static analysis of the code. Furthermore, to keep production costs down, embedded systems resources are usually limited but they must perform within tight deadlines.

When designing a system, we can assign time budgets to the tasks that are not implemented by intelligent guesses based on experience. By doing this, we gain two positive effects. First, the system-level timing analysis can be performed before implementation, thus providing a tool for estimating the performance of the system. Second, the time budgets can be used as an implementation requirement. By applying this approach, we make the design process less ad hoc with respect to real-time performance. In traditional system design, timing problems are first recognized when the complete system or subsystem has been implemented. If a timing problem is then detected, ad hoc optimization will begin, this most surely making the system more difficult to maintain.

The general benefits of CBSE approaches e.g. reusability, flexibility are hard to achieve together with predictability of RT systems [1]. This requires a careful design trade-offs to be made. The non-real time component technologies such as CORBA, JavaBeans etc are not suitable for RT systems development, thus requiring domain specific component frameworks, methodologies, and tool frameworks

OVERVIEW OF PROPOSED RESEARCH WORK

Developing industrial real-time systems is difficult and sets high requirements to system safety and reliability. The short development cycles demand a reliable engineering method, with predictable costs. The state-of-the-art is dominated by an ad-hoc mixture of methods and tools, and system validation is mostly done by extensive testing at the implementation level. However, testing is done already too late in the design process, and bugs may still exist even in well-tested models. In this context, techniques for managing complexity and ensuring critical system properties during design become a necessity.

A promising design approach is to employ a formal component-based development technique. In such an approach, components are introduced as executable software units that can be deployed into a system [4]. One of the key issues of realizing the component-based software paradigm is to ensure that the separately specified components do not conflict with each other when composed, resulting in blocking
the system. A potential solution to this issue is formal modular verification of component-based software via model checking. Besides making the model-checking efficient, an important task is to produce manageable and easy-to-grasp design models for components and their composition. A promising approach is to extract some common behavioral patterns that occur frequently in the design of real-time systems, and represent them in a finite-state-machine like notation [18]. Such notation could let us apply these patterns at high-levels of software development, while simplifying the produced models. The employing patterns in designing component-based systems might also help in documenting the associated software, through pattern-based reverse engineering.

General purpose program design patterns are well-known in the object-oriented design community, for a while now. Nevertheless, in the design of component-based real-time systems, some different aspects might need to be represented in the modeling patterns; for instance, the semantics of SaveCCM components [16] is a read-execute-write semantics, hence a run-to-completion pattern can prove beneficial in the design. Similarly, the reusable modeling of the sequence of visited states during the execution of a component, or reducing the time-wise non-determinism of the real-time component behavior, by providing systematic means to associate a deadline with the behavior, through a pattern, might also help the designer in the modeling phase. Thus, these abstractions of common real-time component behaviors can be formulated as the run-to-completion, history, and execution-time patterns, respectively [18]. The pattern framework can be extended with other design techniques e.g. those of synchronous languages like Lustre, Esterel [8]. These techniques are useful for efficient analysis of real time, embedded systems, and hence can be encoded as useful patterns for design purpose. These patterns can be suitably instantiated by simple transformation mechanisms to existing analysis frameworks, though hiding the latter as needed.

Though independent of specific component models and related tools, the pattern framework is proposed to be validated in modeling the component-based systems using ProCom component model and associated tool environment [15,16]. Further, the proposed research aims to provide formal semantics for ProCom component modeling language. The proposed framework will also be extended with a behavioral language, called REMES [17], for specification of functional, timing, resource consumption of system components. The language will have features from both Statecharts, for expressiveness, and Charon, for analyzability. Further a formal semantics of the component model will be defined to enable formal analysis of components and the system. The research aims at providing an engineering methodology with high-level modeling (with support of modeling in SaveIDE) and practically applicable analysis techniques aiming at verifying system behavior at a higher abstraction level (model checking using Uppaal-Port toolkit). Additionally, the research shall attempt at meeting other challenges stated above namely stepwise refinement, composability, comparability, compositionality etc [8].

The main aspects of the considered research are:
1. Defining intermediate component based modeling languages between architectural descriptions and formal languages
   a) Pattern description language for design, functional and timing aspects,
   b) Behavioral specification language for functional, resource usage

2. Defining formal semantics of component modeling languages

3. Defining transformations between the intermediate modeling languages to semantical frameworks for analysis of composed systems

4. To provide necessary tool support

RELATED RESEARCH WORKS

**BIP Framework approach:** the framework provides an environment for building real time systems with heterogeneous components [7]. An algebraic framework is defined for component based description and analysis of systems. Algebraic laws are defined for component interactions and glue operators. Systems can be described and analysed as a composition of components using glue operators. Architecture of the system is defined in terms of components, interactions, and priorities. Model based engineering: a timed system model is derived from models of application, user requirement, platform (timed), and environment (timed). Following observations may be considered for BIP framework aspects:

- the distinction between 'glue operators' and 'interactions' is not clear. while an interaction is defined between corresponding ports of components, a 'glue operator' seems to be a collection of interactions.
- it is claimed the architecture is independent of component behaviors..but it is not clear as 'priorities' seem to refer to specific actions, transitions of behaviors
- it is required to derive 'sufficient' conditions for formal verification e.g. correctness, schedulability, deadlock-free etc. Thus the approach does not seem scalable for e.g. it is not clear how 'suitable' invariants can be calculated for 'compositional deadlock verification' of systems (using the sufficient condition 'S' and condition 'I and S = false')
- model checking aspect of verification is not very useful..it has more 'deductive' flavour apprently
- the BIP framework appears more inclined towards implementation with issues like heterogeneity and expressiveness rather than formal analysis and scalable 'compositional' analysis methods. thus the framework seems short of much required 'compositionality results' needed
**PTIDES Approach:** 'Timeliness' and 'Concurrency' of computations are identified as the Two challenges for cyber physical systems (CPS). 'Actor oriented' concurrent components with Discrete Event semantics (i.e. distributed DE simulation technology) is proposed. The approach is called 'PTIDES'(pronounced Tides) for 'Programming Temporally Integrated Distributed Embedded Systems' [9]. The concurrent actors exchange 'time stamps' based on 'model time' (not related to physical 'clock' time, though can be binded to 'physical' time only when necessary). Not violating the time constraints implies the 'schedulability' of the system. Sufficient conditions, for e.g. based on static analysis, can be derived for formal analysis of the system. Claimed to be better than 'classical' (due to chandy, Misra) and 'optimal' (due to Jefferson) distributed DE techniques.

Analysis approaches envisaged: exposing modeling errors (classical analysis method with assumption of zero execution time for actors); exposing complexity problems (assuming known WCET for actors and unbounded resources); exposing resource limitations (WCET for actors and scheduling policy over resources);

First analysis method, as stated above, presented using 'causality interfaces' based on min-plus algebra [10]. An execution strategy is derivable from 'causality interfaces' to determine safe conditions to process an event arrived at an actor. 'relevant dependency' relation defined between input ports of an actor. A 'Collapsed graph' can be constructed based on the transitive closure of the dependency relation to compute 'relevant dependencies' between equivalence classes.

Dependency cut for an equivalence class can be calculated which is the minimal but complete set of equivalence classes that needs to be considered to process an event at a given equivalence class.

Some of the observations w.r.t to PTIDES approach are:

- it is an interesting, scalable formal approach
- the approach requires precise calculation of various bounding times e.g. computation time, transmission delay, sensor delay, clock error etc.. which itself apparently looks unreliable as it requires careful analysis of execution time and scheduling policies
- why causality interfaces are defined on 'minimum' model-time delay between ports? why not 'maximum' delays? it is not clear..
- how 'model time delays' can be computed..also what exactly 'model time' refers to is not clear
- It is not clear how formal verification other than timeliness and schedulability can be conducted...e.g. progress properties etc.
- the approach can be extended with behavioral modeling at actors to do model checking of complete system.
**COMDES II approach:** is a generative development methodology and a component-based software framework for distributed embedded control systems with real-time constraints [19]. The adopted methodology allows for rapid modeling and validation of control software at a higher level of abstraction, from which a system implementation in C can be automatically synthesized. To achieve this objective, COMDES-II defines formally various kinds of components to address the critical requirements of the targeted domain, taking into consideration both the architectural and behavioral aspects of the system. Accordingly, a system can be hierarchically composed from reusable components with heterogeneous models of computation, whereas behavioral aspects of interest are specified independently. It emphasizes the following development issues:

- component models of computation (MoC) and the associated modeling techniques that can be used to specify significant characteristics of real-time control systems, following the principle of separation-of-concerns.
- practically applicable analysis techniques aiming at verifying system behavior at a higher abstraction level, e.g. model-checking.
- advanced algorithms and data structures enabling eusability and reconfigurability of components.
- automatic code generation techniques to maximally reduce the manual coding effort, hence minimizing the errors introduced by manual coding.
- proper compilation and configuration techniques, which can be used to automatically synthesize the deployable systems from prefabricated component executables.

COMDES-II employs a hierarchical model to specify system architecture: at the system level a distributed control application is conceived as a network of communicating *actors* (active components). Distributed actors interact transparently with each other by exchanging labeled messages (signals), following an asynchronous producer-consumer protocol known as content-oriented message addressing. At the actor level, an actor is specified as a software artifact containing multiple *I/O drivers* and a single *actor task* (execution thread). I/O drivers are classified as *communication drivers* and *physical drivers*, which are associated with the actor task through a dataflow relationship denoting the exchange of local signals.

Function blocks (FBs) are pure functional components implementing concrete computation or control algorithms, which can be used to specify the system functional behavior. COMDES-II defines four kinds of FBs: *basic*, *composite*, *modal* as well as *state machine* FBs to help specify various kinds of system functionality, in which basic and composite FBs can be used to model continuous behavior (data flow), while state machine and modal FBs describe the sequential system behavior (control flow). Moreover, COMDES-II provides adequate modeling and
implementation techniques integrating heterogeneous data flow and control flow models to realize hybrid (modal continuous) system operation [4].

The timed multitasking model of computation separates the timing behavior of an actor from its functional aspect, which may facilitate timing analysis, since real-time properties can be checked using schedulability analysis, rather than being checked together with the functional behavior (e.g. using timed automata models). The combination of timed multitasking with transparent signal-based communication has resulted in a novel operation model - Distributed Timed Multitasking (DTM) [10], which is one of the distinctive features of COMDES-II. It provides for jitter-free execution of periodic distributed actor transactions, whereby transaction input and output actions are executed at precisely specified time instants on the time axis.

**Other approaches:** The other prominent methodologies for embedded system design are based on modeling languages such as the Unified Modeling Language (UML) and Architecture Analysis and Design Language (AADL) and go a step beyond implementation independence [3]. They attempt to be generic not only in the choice of implementation platform, but even in the choice of execution and interaction semantics for abstract system descriptions. This leads to independence from a particular programming language, as well as to an emphasis on system architecture as a means of organizing computation, communication, and resource constraints.

Much recent attention has focused on frameworks for expressing different models of computation and their interoperation [10]. These frameworks support the construction of systems from components and high-level primitives for their coordination. They aim to offer not just a disjoint union of models within a common metalanguage, but also to preserve properties during model composition and to support meaningful analyses and transformations across heterogeneous model boundaries.

**SOME RESEARCH CHALLENGES**

It is important to derive a mathematical basis [11] for systems modeling and analysis that integrates both abstract-machine models and transfer-function models. Such a theory could unite two sets of practices [7,8]: those from critical systems engineering that guarantee hard requirements and those from best-effort systems engineering that optimize performance. From there, the theory, methodologies, and tools must encompass heterogeneous execution and interaction mechanisms for system components, provide abstractions that isolate the design sub-problems requiring human creativity from those that can be automated, scale by supporting compositional, correct-by-construction techniques, and ensure the robustness of the resulting systems [8]. A practical methodology for embedded systems design needs to scale, and overcome the limitations of current algorithmic verification and synthesis techniques. One route for achieving scalability is to rely on compositionality and noninterference rules which require only light-weight analyses of the overall system architecture.
Related Conferences & Forums:

SEFM: Software Engineering and Formal Methods

- Scalable formal methods, software specification, verification and validation, component-based development, formal models for service-oriented computing, model checking for software and hardware systems, real-time, hybrid and embedded systems, safety-critical and fault-tolerant systems software architectures and their description languages, light-weight formal methods, CASE tools and tool integration, applications of formal methods and industrial case studies

ICFEM: International Conference on Formal Engineering Methods

- Formal methods for object and component systems, Applications in model-driven and service-based architectures, Abstraction and refinement, Tool development and integration for system design and verification, Integration of formal verification tools in CASE tools, Techniques for specification, verification and validation, Techniques and case studies for correctness by construction, Experiments involving verified systems, Applications in real-time, hybrid and critical systems, Development methodologies with their formal foundations

COMPSAC: IEEE International Computer Software and Applications Conference

- Topics of interest include but are not limited to requirement analysis, co-analysis/co-design, modelling, development, testing, measurement, verification, validation, performance, autonomy, safety, security, and dependability constraints

Concurrency mailing list: http://homepages.cwi.nl/~bertl/concurrency/


REFERENCES


13) John Håkansson, Component based Modelling of Embedded Real-time systems, Chapter 03, Ph.D thesis draft, 2008 (internal)

14) J. Carlson, J. Håkansson, and P. Pettersson, SaveCCM: An analysable component model for real-time systems, FACS 2005


16) Severine Sentilles et.al, Save-IDE – An integrated development environment for building predictable component-based embedded systems, 2008

