Supporting Model-driven Development of Component-based Embedded Systems with Cadena

Adam Childs, Xianghua Deng, Matthew B. Dwyer, Jesse Greenwald, John Hatcliff,
Prashant Kumar, Georg Jung, Venkatesh Ranganath, Robby, Gurdip Singh
Department of CIS
Kansas State University
dwyer@cis.ksu.edu

ABSTRACT

Developers of modern distributed, real-time embedded systems are increasingly employing component-based middleware frameworks to cope with the ever increasing size and complexity of mission requirements. Frameworks, such as CORBA and its component model (CCM), raise the level of abstraction at which systems are programmed by directly supporting certain essential non-functional requirements of these applications, such as distribution, and through add-on capabilities, such as services that provide event-driven execution. We describe how such frameworks can be used as a foundation for providing even higher-levels of abstraction in system development that can be leveraged throughout the entire software development process. Specifically, we outline the key goals and capabilities of Cadena, an integrated environment that supports the model-driven development of CORBA-based real-time embedded systems.

1. MODERN EMBEDDED SOFTWARE

As the processing power available in embedded platforms continues to increase, so too does the demand for increasingly capable software to meet challenging mission requirements. Functionality that once was "off loaded" to non-embedded computing nodes is now routinely being deployed in embedded devices. The problems of developing systems that reliably meet stringent timing requirements were hard enough, but now embedded systems developers are confronting the same problems of scale, complexity and distribution that trouble developers in more mainstream application domains. Consequently embedded system developers are adopting standardized component-based development frameworks, such as CORBA, to meet those challenges and adapting them to address timeliness requirements (e.g., [1]). We believe that such frameworks can be enhanced to provide more effective support for the development of highly-reliable embedded software.

To make our discussion more concrete, consider the "push" model of computation used in the BoldStroke middleware [4] which is an adaptation of CORBA. In such a model, a component subscribes to events that signal the availability of data that comprise its inputs, when triggered a component access that data (either from the event payload or via component method calls), it then perform internal calculations and if new values result publishes an event indicating their availability.

For such systems, it is often convenient to divide development into *component development*, where generic and application specific functionality is implemented and packaged with CCM Interface Definition Language (IDL) defined interfaces, and *component integration*, where potentially large numbers of components are instantiated and assembled to fulfill the overall system requirements. This division has the advantage that component development can be reduced, in the best case, to implementation of small simple sequential blocks of code. Unfortunately, component integration remains extremely difficult.

CCM IDL and CCM-compliant middleware, such as OpenCCM [2], provide only a limited form of modeling (i.e., defining the input/output structure of component types), yet they are attractive since they automate significant amounts of platform specific coding. To develop a complete working system, however, additional details must be provided as code, such as, component instantiation, event subscription, and component method synchronization. These tasks fall to the component integrator who requires an understanding of global system behavior to define those details. Unfortunately, despite the component nature of middleware frameworks, global reasoning requires consideration of assemblies of component instances; modular reasoning is not well supported. Perhaps surprisingly in event-driven systems, such as the push model described above, even simple control flow relationships, that are syntactically apparent in many system descriptions, are obfuscated by the inversion of control provided by the middleware. This makes it very difficult to to determine the functional properties of a system much less properties related to real-time, data-coherence, correct synchronization, etc. *Automated analysis of global properties of component assemblies must be available early in the development process for effective component integration.*

It is sometimes necessary to compromise natural component and framework abstractions in order to achieve essential system correctness properties or desired levels of performance. For example, when a system contains some instances of a component type that require synchronization

Specification and Verification of Component-based Systems, Sept. 2003

and others that do not, this non-functional aspect forces developers to define variants of the common functional component interface. Fast-path middleware optimizations can often be performed when knowledge about component assembly (e.g., component co-location) is available. Current approaches that delay optimization to run-time miss opportunities for even greater performance that could be achieved by static exploitation of information about component assemblies that is not available in existing IDL. *Component and system modeling notations must be enriched to include additional semantics to enable effective analysis and to increase the scope and quality of system synthesis.*

In this context, we have developed Cadena an integrated environment intended to support the definition, validation and synthesis of component-based, distributed, real-time, embedded software from high-level structural and behavioral models [3].

2. CADENA GOALS AND CAPABILITIES

The primary goal of Cadena is to address the problems encountered during component integration by (*i*) providing developers with feedback on system correctness properties early in the development process, (*ii*) enriching the existing synthesis technologies in middleware frameworks to generate more of the application code-base, and (*iii*) exploiting information about component assemblies to drive automatic performance optimization.

Our strategy is to exploit existing IDL as a basis for layering additional light-weight specification forms. The intent is to provide a means of balancing developer investment with accrued benefit to help address the high entry-barrier of using formal notations that typically stifles their adoption. Specifically, we have developed a suite of specification forms that describe component *instantiation* (i.e., naming and parameterization of component types), *assembly* (i.e., defining instance event subscriptions), *rate* (i.e., defining instance execution priority), *distribution* (i.e., defining an instance's location within the system), *dependences* (i.e., defining dependencies between component inputs and outputs), *states and transitions* (i.e., defining component attributes that persist across method executions and transitions of attributes achieved by execution of component methods and event handlers), and *synchronization* (i.e., defining synchronization policies for component methods) Our approach allows related specifications to be viewed as refinements, for example, one transition system description may refine another or it may refine a dependence specification. Using these forms, a developer may start with IDL and then selectively add focused semantic information to enable specific kinds of analysis and synthesis.

Associated with each of these specification forms is an analysis capability including: *event dependence checking* (i.e., using dependence and state transition forms to answer queries based on forward/backward slicing/chopping, and to detect anomalous event sequences such as cycles and published events without subscribers), *design advice* (i.e., heuristic-driven algorithms that use structural properties of component assemblies to generate candidate assignments for instance rate and location assignments), and *state-space search* (i.e., an abstract parameterizable semantic model of middleware and environment behavior is combined with state transition forms to answer queries about the sequencing of program actions and reachable component states). As more forms are layered onto a system description the analyses consider their composition, for example, state-space search will exploit specified or generated component rate information to eliminate searching infeasible system schedules. When a form is absent the analyses make safe assumptions about the unspecified behavior (e.g., that a component may run at any rate). Additional forms of analysis, such as timing and schedulability analysis, are being integrated into Cadena through external tool APIs.

The goal of these analyses is two-fold: to provide integrators with feedback about system properties and to drive high-quality system synthesis. Cadena is currently capable of synthesizing system configuration code that encodes event subscription, rate and distribution information for BoldStroke middleware. Ongoing work is enriching these capabilities to add synthesis of component code from state transition and synchronization policy specifications. This holds the promise of further simplifying component development by reducing it to straight-line sequential code with calls to well-understood library routines to achieve application specific data transformation. Future work includes the definition of customization APIs in middleware frameworks that enable model-driven optimization of execution paths within the middleware. While individual examples of such optimizations, for example replacing publish-subscribe mechanisms with method calls for co-located components, have proven to be very effective, we are working towards more general support for middleware customization.

3. CURRENT STATUS AND ONGOING WORK

Cadena is under active development, but we are making binary releases available to community at `http://cadena.projects.cis.ksu.edu`. Releases include a tutorial and a variety of example system models. A wide variety of system description and visualization forms are currently supported as are a suite of analyses. Support for system generation is via integration with OpenCCM.

Work on Cadena is supported by the U.S. Army Research Office (DAAD190110564) and by DARPA/IXO's PCES program (AFRL Contract F33615-00-C-3044). As part of these efforts, we are applying Cadena to model, analyze and generate systems that are representative of actual mission computing systems for fighter aircraft in terms of both their size (more than six hundred components) and complexity (thousands of event publications per scheduling period).

4. REFERENCES

[1] B. Doerr and D. Sharp. Freeing product line architectures from execution dependencies. In *Proceedings of the Software Technology Conference*, May 1999.

[2] GOAL. The OpenCCM platform. `http://corbaweb.lifl.fr/OpenCCM/`, 2002.

[3] J. Hatcliff, W. Deng, M. Dwyer, G. Jung, and V. Prasad. Cadena: An integrated development, analysis, and verification environment for component-based systems. In *Proceedings of the 25th International Conference on Software Engineering*, 2003.

[4] D. Sharp. Object oriented avionics software flow policies. In *Proceedings of the 18th AIAA/IEEE Digital Avionics Systems Conference*, Oct. 1999.