"Real – Time Software Components for Embedded Systems"

Dr. Ahmedi Azra

Research Scholar

Abstract – In this paper we discussed real-time software components for embedded systems. Real-time systems are computer systems that monitor, respond to, or control an external environment. This environment is connected to the computer system through sensors, actuators, and other input-output interfaces. It may consist of physical or biological objects of any form and structure. Often humans are part of the connected external world, but a wide range of other natural and artificial objects, as well as animals, are also possible.

Keywords: Embedded real-time systems; Component model; Non-functional properties; Separation of concerns

----------♦----------

INTRODUCTION

The computer system must meet various timing and other constraints that are imposed on it by the real-time behavior of the external world to which it is interfaced. Hence, the name is *real time*. Another name for many of these systems is *reactive systems*, because their primary purpose is to respond to or react to signals from their environment. A real-time computer system may be a component of a larger system in which it is embedded; reasonably, such a computer component is called an *embedded system*.

Applications and examples of real-time systems are ubiquitous and proliferating, appearing as part of our commercial, government, military, medical, educational, and cultural infrastructures. Included are

- vehicle systems for automobiles, subways, aircraft, railways, and ships
- traffic control for highways, airspace, railway tracks, and shipping lanes
- process control for power plants, chemical plants, and consumer products such as soft drinks and beer
- medical systems for radiation therapy, patient monitoring, and defibrillation
- military uses such as firing weapons, tracking, and command and control
- manufacturing systems with robots
- telephone, radio, and satellite communications
- computer games
- multimedia systems that provide text, graphic, audio, and video interfaces
- household systems for monitoring and controlling appliances
- building managers that control such entities as heat, lights, doors, and elevators

Component based development has proven itself to have significant benefits in the enterprise IT and web-based environments; complex applications are quickly created and very few, if any, are created from scratch – they all leverage a rich set of third-party open-source or commercial components. Beyond object-oriented language support, component models address all phases of the software lifecycle and standardize software abstractions to the point where – without any ad-hoc conventions or schedule coordination among the participants – interfaces defined by one company can be implemented by a second and used by a third. Moreover, tools that leverage the additional structure imposed by this standardization facilitate the development and (re)use of components,

further accelerating the creation of applications assembled from components created by third-parties.

RTSC OVERVIEW

The RTSC tools and component model have enjoyed continuous development since 2000 by a small group of senior embedded software developers. Since 2004, the DSP/BIOS 5.x RTOS – created using the RTSC tools – has shipped along with the RTSC tools to ensure that any development system that included DSP/BIOS could consume components (called packages) created by any other development group. Today, internal groups within Texas Instruments regularly (re)build, test, and deploy hundreds of RTSC packages. Many of these packages are used worldwide by thousands of developers both inside and outside Texas Instruments.

- **DSP/BIOS 5.x** – one of the most popular embedded RTOS's – is deployed as a bundle of more than 56 packages,
- **Codec Engine** multi-media middleware runtime (which requires DSP/BIOS) is an independently deployed bundle of more than 21 packages,
- a wide variety of video, imaging, speech, and audio codec's – developed by both Texas Instruments and its third parties – are delivered as a packages, and
- the RTSC toolset itself is delivered as a bundle of over 125 packages

The fact that developers have been using DSP/BIOS 5.x without realizing that it is, in fact, a collection of RTSC components illustrates one of the strengths of the RTSC model: consumers of RTSC components can easily integrate them without converting their entire application into components.

The RTSC tools currently provide basic support for the entire software development cycle: Install Develop, Debug, and Deploy.

- Install:
  - package selection, compatibility checks, and side-by-side installation
- Develop:
  - side-by-side multi-target build with managed tool chains for both cross and native compilers
  - tool chain-independent package build specifications
  - component configuration and assembly tool
  - document generation from component specifications
- Debug:
  - component-specific views of internal data structures
  - component compatibility checking
- Deploy:
  - component packaging tools
  - on-device real-time logging and diagnostics to monitor system activity

Among the various solutions proposed to that end, the adoption of Model-Driven Engineering (MDE) Schmidt (2006) has fared rather well by measure of interest and success. Evidence collected in domain-specific initiatives (cf. e.g., (Bordin and Vardanega, 2007, Panunzio and Vardanega, 2007 and Bordin et al., 2008)) shows that the higher level of abstraction in the design process facilitated by MDE allows addressing non-functional concerns earlier in the development, thereby enabling proactive analysis, maturation and consolidation of the software design. Moreover, the automation capabilities of the MDE infrastructure may ease the generation of lower-level design artifacts and enable the automated generation of source code products of certain quality.

ISO/IEC/(IEEE) (2007) defines an architecture as composed of: (a) the fundamental organization of a system embodied in its components; (b) their relationships to each other, and to the environment; and (c) the principles guiding its design and evolution. On that basis, Panunzio and Vardanega (2013) regards the concept of software reference architecture as proceeding from: (i) a component model, to design the software as a composition of individually verifiable and reusable software units; (ii) a computational model, to relate the design entities of the component model, their non-functional needs for concurrency, time and space, to a framework of analysis techniques which assures that the architectural description is statically analyzable in the dimensions of interest by

construction; (iii) a programming model, to ensure that the implementation of the design entities obeys the semantics, the assumptions and the constraints of the computational model; (iv) a conforming execution platform, which actively preserves at run time the system and software properties asserted by static analysis and it is able to notify and react to possible violations of them.

Although real-time embedded systems are increasingly being developed with object-oriented languages and techniques, to enable the same level of cross-company reuse and rapid application development, a component model and supporting tools are needed. However, existing enterprise models (such as JavaBeans, .NET, Corba, etc.) do not address the unique challenges of embedded systems:

- *Embedded platforms are extremely cost and power sensitive :* to minimize cost and power consumption, a wide variety of CPUs, peripherals, and memories are employed with limited code space and MIPS capacity.

- *Embedded software must be "optimal":* to work within the constraints of small memory and relatively slow clock rates necessitated by the cost and power constraints, software must be as small and fast as possible.

- *Existing software is predominantly written in C and assembly language:* neither standard definition of interfaces nor a common runtime that enables multiple implementations of an interface within a single application exists.

- *No standard C/C++ compiler tool chain exists for all devices:* while GCC supports many CPUs, to achieve the necessary performance from their "portable" ANSI C code bases, developers must leverage C/C++ compilers from the device manufactures that achieve "optimal" performance for their devices.

Several component models have been created to meet these challenges, but these models and their tool chains are often tied to a specific compiler, embedded operating system, embedded hardware platform, or host development platform. In addition, the more sophisticated models can't be scaled down to support popular but resource constrained devices such as an Intel 8051 or a Texas Instruments MSP430; for example, Corba all but requires a C++ runtime but – because of device memory constraints – no practical C++ support exists for the MSP430. As a result, embedded developers can rarely leverage these models and can't afford to invest the time required to learn them. Components created for these models can only be used in a limited number of embedded platforms, defeating the opportunity to reuse these components or the skills required to create them in more than just a few closely related projects.

THE RTSC MODEL

Sometimes through heroism you can make something work. However, understanding why it worked, abstracting it, making it a primitive is the key to getting to the next order of magnitude of scale. – Robert Calderbank

The RTSC model and tools enable development of components written in C using *any* compiler tool chain on *any* development host for *any* embedded platform. These components can be configured, assembled, and optimized for use within any embedded real-time system. By focusing on *design-time* rather than on runtime component assembly, the RTSC model and tools enable many of the component-based benefits to scale down to even the most resource constrained embedded system while leveraging existing C/C++ code bases and tool chains.

The RTSC tools, developed over a period of 7 years, are already in use by several Texas Instruments (TI) development groups and have been used to produce "mass market" products such as the DSP/BIOS Real-Time Operating System and the Codec Engine multi-media middleware framework. While these products enjoy the benefits of not having to reinvent the capabilities provided by the RTSC tools, the value of these tools and the motivation to create new tools increases dramatically as adoption of RTSC increases. However, wide-spread adoption is only possible if the model and base tooling are open and freely available.

SCOPE

The goal for the RTSC project is to refine and standardize the core RTSC component model and foundational tools in an effort to bring component-based development advantages to *all* embedded C/C++ developers. The elements included in this project are listed in the Core Architectural Elements section below.

By making this core infrastructure open, extensible, and freely available, we expect to seed additional projects that provide

- More sophisticated tools for component development: unit test frameworks, refactoring tools, etc.
- Integration with other popular embedded tools and languages: UML, Doxygen, static checking tools (e.g., Coverity Prevent or Klockwork), etc.
- alternative or domain-specific component composition tools; e.g., a GEF based tool to create an application from existing components or a multi-core component development environment such as Zeligsoft's CE 3.0 product
- rich visualization of component-based applications: graphical representations of the relationships among constituent components, Dependency Structure Matrixtools, etc.
- extensions of existing component-based tools enjoyed by the Java developer to support RTSC components

It is *not* a goal of this project to create the tools described above, rather the goal is to provide the common foundation to enable the creation of these more advanced capabilities by other groups. We want nothing less than a robust component-based development platform suitable for *any* embedded system built atop Eclipse.

Each of these projects benefits from a standard underlying component model and will, if we are successful, bring a complete set of modern component-based tools to all embedded C/C++ developers. In addition, these projects will help shape the roadmap of the core RTSC project by adding new requirements or uncovering new use-cases that need to be supported.

CONCLUSION:

In this paper we found that the RTSC component model centers around three top-level concepts: *modules*, *interfaces*, and *packages.* Roughly speaking, modules correspond to Java or C++ classes, interfaces correspond to Java interfaces, and packages correspond to Java jars. Unlike Java, however, RTSC components provide code for two distinct environments: development hosts (with "unlimited" resources) and embedded runtime platforms (with very limited resources). It is this ability for components to operate in and be "configured" on the development host that allows them to scale their runtime requirements to a level appropriate for each embedded system in which they operate.

REFERENCES:

- Schmidt, 2006, D.C. Schmidt,Model-driven engineering, IEEE Comput., 39 (2) (2006), pp. 25–31
- Bordin and Vardanega, 2007,M. Bordin, T. Vardanega,Correctness by construction for high-integrity real-time systems: a metamodel-driven approach,Proceedings of the 12th International Conference on Reliable Software Technologies – Ada-Europe (2007)
- Panunzio and Vardanega, 2007,M. Panunzio, T. Vardanega,A metamodel-driven process featuring advanced model-based timing analysis,Proceedings of the 12th International Conference on Reliable Software Technologies – Ada-Europe (2007)
- ISO/IEC/(IEEE), 2007, ISO / IEC / (IEEE), Systems and Software Engineering – Recommended Practice for Architectural Description of Software-Intensive Systems, ISO/IEC 42010 (IEEE Std) 1471-2000,(2007)
- Panunzio and Vardanega, 2013,M. Panunzio, T. Vardanega,On software reference architectures and their application to the space domain,13th International Conference on Software Reuse (2013), pp. 144–159
- Bordin et al., 2008,M. Bordin, M. Panunzio, T. Vardanega,Fitting schedulability Analysis theory into model-driven engineering,Proceedings of the 20th Euromicro Conference on Real-Time Systems (2008)

Web links-

- http://eclipse.org/proposals/rtsc/
- http://www.sciencedirect.com/science/article/pii/S0164121214001381
- http://www.cs.uni.edu/~mccormic/RealTime/what.html