CARTS: A Tool for Compositional Analysis of Real-Time Systems

Linh T.X. Phan Jaewoo Lee Arvind Easwaran Vinay Ramaswamy Sanjian Chen Insup Lee Oleg Sokolsky
Department of Computer and Information Sciences, University of Pennsylvania
Email: {linhphan, jaewoo, arvinde, vinayr, sanjian, lee, sokolsky}@cis.upenn.edu

I. Introduction

As real-time embedded systems are increasingly complex, integration becomes a great challenge in their design and development. Managing complexity of the system design is therefore essential for high-assurance and cost-effective development. Component-based design has consequently been developed and gained its importance over the years as a powerful technique for complexity management. In this design paradigm, a large complex system is first decomposed into smaller and simpler components – which are developed independently – before recomposing them into a complete system using interfaces that abstract away their internal complexities.

To facilitate component-based design, given a component, one needs to be able to compute the component interface - an appropriate abstraction of the component's resource requirement. This resource interface can be computed either directly from the component's workload or by composing the interfaces of the subcomponents. Accurate and efficient interface generation/composition techniques and tools are therefore crucial for the component-based design of the system.

To meet the growing needs, we have developed CARTS (Compositional Analysis of Real-Time Systems) as a platform-independent tool that automatically generates resource interfaces needed for the compositional analysis of real-time systems. Our tool is built on top of several existing theoretical compositional analysis frameworks for real-time systems [2], [4], [6], [7]. Besides supporting standard schedulers, such as Rate Monotonic (RM) and Earliest Deadline First (EDF), it generates both *periodic* and *explicit deadline periodic* resource interfaces. The tool also comes with a friendly GUI and a rich set of tool features that allow designers to specify and analyze a wide variety of systems at ease. At the same time, it is also accompanied by a lightweight command-line option that enables our tool to be integrated with other existing toolchains.

II. Theoretical foundation underlying CARTS

In a hierarchical scheduling framework, the system is partitioned into a tree of components that are scheduled in a hierarchical manner. Each internal node of the tree represents a *composite component*, whose children are its sub-components. Each leaf represents an *elementary component*, which is a finite set of tasks in the system.

Figure 1 shows a composite component C made of two elementary components C_1 and C_2, which are scheduled under EDF. Component C_1 consists of two tasks T_1 and T_2, which are scheduled under EDF. Component C_2 consists of two tasks T_3 and T_4, which are scheduled under RM.

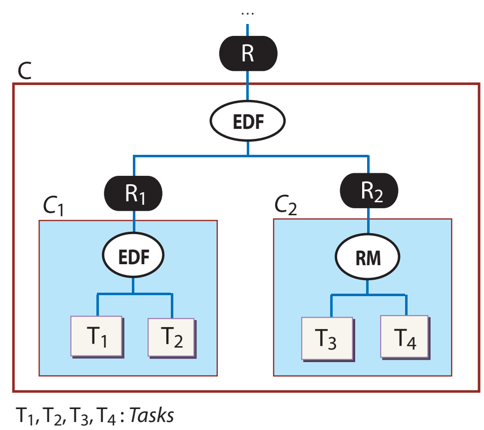

Fig. 1. Hierarchical scheduling of a system component.

The compositional analysis of the example component C is a two-step process: (1) compute the resource interface R_1 (resp. R_2) of the component C_1 (resp. C_2) based on the resource demands of the tasks in the component; (2) compute the resource interface R by composing R_1 and R_2. The resource interface R is then composed with the interface of other components to form the interface of the upper-level component.

Task model	(p,e)	(p,e,o,j,d)	(p,e,x)
Resource model			
Periodic (Π,Θ)	EDF, RM	X	RM
EDP (Π,Θ,Δ)	EDF, RM	EDF, RM	X

TABLE I
Models and scheduling policies supported by CARTS.

The CARTS core engine runs several algorithms for computing the interface of any component or a hierarchy of components that are scheduled under RM or EDF. Table I summarizes the task models and the scheduling policies for the components, and their corresponding resource interface models supported by CARTS. We briefly describe these models below.

Task models. CARTS supports three different variants of periodic task models, including: (i) strictly periodic task with deadline equal to period [7], defined by $T = (p,e)$;

(ii) periodic task with jitter and offset [3], defined by $T = (p, e, o, j, d)$; and (iii) strictly periodic tasks with resource sharing [1], defined by $T = (p, e, x)$. Here, p, e, o, j, d, x denote the period, worst-case execution time, offset, jitter, relative deadline, and the worst-case execution time of the task in a critical session, respectively.

Resource interface models. A component interface produced by CARTS engine can be given as either a *periodic* or an *explicit deadline periodic* resource model. A periodic resource model [7] is defined by $\Gamma = (\Pi, \Theta)$ where Π is the period and Θ is the resource allocation time $(0 < \Theta \leq \Pi)$. Semantically, each periodic resource model $\Gamma = (\Pi, \Theta)$ provides Θ units of resource in every Π time units. An explicit deadline periodic (EDP) [2] resource model is characterized by $\Omega = (\Pi, \Theta, \Delta)$ where Π and Θ are defined as in the periodic resource model, and Δ is the explicit deadline $(0 < \Theta \leq \Delta \leq \Pi)$. Semantically, EDP is similar to the periodic resource model, except that it must provide Θ resource units within Δ time units.

III. The Design of CARTS

Figure 2 depicts the different components of the tool and their interactions. At the back-end, the *Scheduling Tree* class contains the reference to the root component of the system. The *Scheduling Algorithms* package implements the CARTS core engine. The *XML Interpreter* class contains methods for parsing the XML input file and building *Scheduling Tree* instances. Finally, the *DBF/SBF Graph Generator* class is responsible for drawing the DBF and SBF of the components and their interface models. The rest of the components constitute the GUI front-end, which is detailed below.

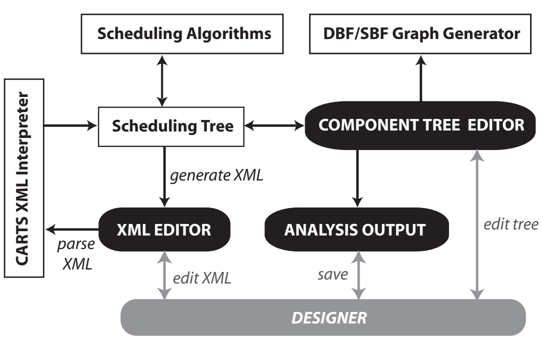

Fig. 2. CARTS system architecture.

Figure 3 shows an overview of the tool GUI. As shown in the figure, CARTS provides two mechanisms for specifying a system model: via the Component Tree Editor (the top left window) or via the XML Editor (the top right window). The component tree editor is equipped with various features that enable designers to easily add, modify and remove components/tasks in a system, using either the top menu or the context menu. The XML editor allows an input XML file description of the system following a simple CARTS XML template. Further, one can also conveniently convert between these two descriptions with a click of the convert button.

Depending on the input component task model, one can apply the different algorithms for generating the component's interface (as outlined in Table I) via either the component's context menu or the top menu. The computed interface will then be displayed in the Analysis Output window as an interface tree, where each node gives the resource interface of the corresponding component in the input component tree. The output results can also be saved as an XML file in a similar format as the input XML description, thereby allowing further processing and/or interfacing with other tools. Additionally, the tool also supports visualization of the demand bound functions of the components and the supply bound functions of the computed resource interfaces.

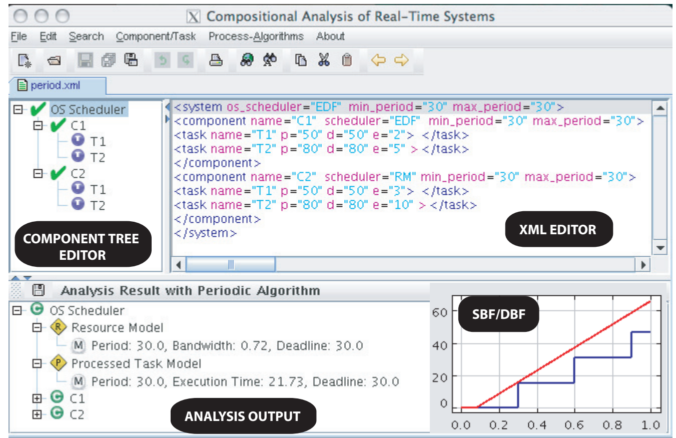

Fig. 3. CARTS graphical user interface.

IV. Concluding Remarks

This paper demonstrates CARTS, a compositional analysis tool for real-time systems. We presented an overview of the underlying theoretical foundation and the architecture design of the tool. CARTS is open source and available for free download at `http://rtg.cis.upenn.edu/carts/`.

We plan to extend the current CARTS tool to support: (i) more general task models (e.g., arrival functions [8], tasks with dependency) and SBF-based resource models (e.g., service functions [8]); (ii) other common scheduling policies (e.g., TDMA, Round Robin); and (iii) multi-mode systems [5]. Finally, we will investigate and incorporate into our toolset compositional analysis techniques for real-time components scheduled on multiprocessor platforms such as [6].

References

[1] Moris Behnam, Insik Shin, Thomas Nolte, and Mikael Nolin. Sirap: a synchronization protocol for hierarchical resource sharingin real-time open systems. In *EMSOFT*, 2007.

[2] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *RTSS*, 2007.

[3] A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal. A compositional scheduling framework for digital avionics systems. *RTCSA*, 2009.

[4] A. Easwaran, I. Shin, O. Sokolsky, and I. Lee. Incremental schedulability analysis of hierarchical real-time components. In *EMSOFT*, 2006.

[5] L.T.X. Phan, I. Lee, and O. Sokolsky. Compositional analysis of multi-mode systems. In *ECRTS*, 2010.

[6] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *ECRTS*, 2008.

[7] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–39, 2008.

[8] Ernesto Wandeler and Lothar Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *RTAS*, San Jose, USA, 2006.