Construction a Real-Time Component for Developing Embedded Real-Time Systems

Shourong Lu¹, Qichun Jin²

¹,² School of Mechanical & Electrical Engineering, Wuxi City College of Vocational Technology, 12 Qianou Road, Huishan District, Wuxi City, Jiangsu Province, 214153, P.R.China

E-mail: ¹shouronglu@hotmail.com, ²694146000@qq.com

Abstract. A real-time component for developing embedded real-time systems is constructed by defined real-time contracts which integrate with component-based techniques. Taking into account the real-time non-functional requirements, the defined real-time contracts are assigned to both static specification (components) and dynamic specification (state machine and sequence diagram). Based on the enhanced specification, it is useful and easier to design embedded and real-time systems.

1. Introduction

The correctness of a real-time system does not only depend on the logical result of its computations, but also on the time at which these results are generated [9]. The properties of a real-time system have strong implications on the requirements, for instance, strict timing constraints, concurrency, high reliability and, application-specific design. These display non-functional characteristics. Real-time systems are also known as reactive systems consisting of several components which are designed to interact with one another and with the systems’ environments. The output of one component can be used as the input to another one. In addition, real-time systems often work in embedded ways and most embedded systems have real-time processing needs. A real-time system is typically constructed of a number of resources (e.g., processors) and a number of concurrent tasks. They are designed to fulfill a number of timing constraints. If a resource is shared by a number of concurrent tasks, a scheduler is needed.

Currently, component-based techniques appear to be an interesting approach in developing embedded and real-time systems on aspects of rapid development times, the ability to re-use existing components, and the ability to compose sophisticated software [1][3], but the component-based techniques are originally not suitable to develop embedded and real-time systems due to they do not address issues such as timeliness, non-functional and predictability properties. Therefore, it is useful and significance with combination of real-time contracts [2][6] and component-based techniques for meeting the non-functional requirements of embedded and real-time systems.

The paper is organized as follows, section 2 constructs a real-time component for real-time tasks, the behavior specification of the real-time component is addressed in section 3, and section 4 concludes the real-time component.

2. Define a real-time component
Real-time tasks are characterized by periodic, aperiodic, and sporadic as shown in Fig 1. Its real-time component can be defined [4] as follows:

![Figure 1. Real-time tasks](image)

2.1. Periodic Component

Periodic component is defined to represent cyclic activities with predefined periods of time. To realise periodic activities with regular activation rates, operational constraints have been defined. The activation of a periodic component must finish its execution before a specified deadline \( D \). Its re-activation depends on the passage of a predefined period \( P \) as shown in Fig. 1 (a,b). The periodic thread of control of a periodic component is represented by its mechanisms and contracts. Formal abstract semantics of periodic components are defined as follows:

A periodic component \( C_{\text{Per}} \) is a structure with the properties:

\[
C_{\text{Per}} = (P_{\text{Per}}, O_{\text{Per}}, r_{\text{Per}}, p_{\text{Per}}, M_{\text{Per}}, \text{Contract}_{\text{Per}})
\]

\( P_{\text{Per}} \) is a set of ports. Periodic component interfaces are attached to ports.

\( r_{\text{Per}} \) is a set of required interfaces. The inputs or events may cause the activation of a periodic component’s operation after the passage of a period of time.

\( p_{\text{Per}} \) is a set of provided interfaces. The periodic component produces an output at a certain time during its execution.

\( O_{\text{Per}} \) is a set of operations which may change the state of the component over time. A mandatory operation that must be defined in a periodic component is the real-time mechanism which controls all actions of the component.

\( M_{\text{Per}} \) is a set of real-time operations. In general, a periodic component and its thread are characterised by its real-time parameters. Through the use of these parameters the periodic activities can be constrained. The operation of a periodic component is characterised by the parameters:

- **Start-time** \( S \): the instant at which the operation of a periodic component starts its execution,
- **Finish-time** \( F \): the instant at which the operation of a periodic component finishes its execution,
- **Deadline** \( D \): the latest point of time at which the operation of a periodic component must be complete to avoid damage to the real-time system and its environment.
Period $P$: the period with which the operation of a periodic component is regularly activated.

*Contract*$_{per}$ supports the periodic activities of a periodic component subject to the following constraints.

$S_i < F_i \leq D_i$: this constraint relates to the start and finish time of the periodic component’s operation. It denotes that the start time must be less than the finish time, respective of its deadline. The finish time must be less or equal to a specified upper-bound time.

$D_i \leq P$: this constraint assumes that the upper-bound time (i.e., the deadline $D$) is always less or equal to the periodic activation rate $P$ of the component.

$S_{i+1} = S_i + P$: this constraint defines the next activation time $S'$ of the periodic component, which is computed by adding the start-time $S$ and the constant period $P$.

2.2. Aperiodic component

Aperiodic component activations are not regular in comparison to periodic components. They execute predefined actions in response to the occurrence of aperiodic, asynchronous events (see Fig. 1 (c,d)). An aperiodic component $C_{aper}$ is defined with the properties:

$C_{aper} = (P_{aper}, O_{aper}, I_{aper}^{required}, I_{aper}^{provided}, M_{aper}, Contract_{aper})$

$P_{aper}$ is a set of ports. Aperiodic component interfaces are attached to ports.

$O_{aper}$ is a set of operations which may change the state of the component over time depending on the occurrence of an initiating event. The occurrence of an event triggers the corresponding mechanisms, in which all actions provided by the aperiodic component are defined.

$I_{aper}^{required}$ is a set of required interfaces. The input is a finite set of initiating, internal or external events $E = \{ev_1, ..., ev_n\}$, causing the activity of the aperiodic component’s operation over time.

$I_{aper}^{provided}$ is a set of provided interfaces. The outputs of the component may produce a set of outputs over time.

$M_{aper}$ is a set of mechanisms. The operation of an aperiodic component is characterised by mandatory attributes marking the beginning with a StartTime and a maximum response time of component executions ($MaxRT$), where $MaxRT$ denotes a soft deadline.

*Contract*$_{aper}$ constrains the activation time of the aperiodic components. Activation times of aperiodic components are constrained as follows.

$S_{i,k} < S_{i,k+1} \Rightarrow S_{i,k} < MaxRT \leq S_{i,k+1}$ where

$S_{i,k}$ denotes the $k^{th}$ activation time of component i, and $S_{i,k+1}$ its $(k + 1)^{th}$ activation time.

The constraint above means that the $(k+1)^{th}$ activation time is greater than the $k^{th}$ one of the aperiodic component’s real-time operation. Furthermore, the real-time operation’s maximum response time is defined to be between the activation times $S_{i,k}$ and $S_{i,k+1}$.

Sporadic component has been introduced in order to specify sporadic activities. It acts as active component and possesses an independent thread of control. A sporadic component executes sporadic activities with a predefined minimum arrival time ($T_{min}$). Its activation is triggered by the occurrence of an event, whose arrival time is known for a sporadic component in contrast to aperiodic components.

2.3. Sporadic Component

Sporadic component has been build in order to specify sporadic activities (see Fig. 1. (e,f)). It acts as active component and possesses an independent thread of control.

A sporadic component $C_{spro}$ is defined with the properties:

$C_{spro} = (P_{spro}, O_{spro}, I_{spro}^{required}, I_{spro}^{provided}, M_{spro}, Contract_{spro})$

$P_{spro}$ is a set of ports. Sporadic component interfaces are attached to ports.

$O_{spro}$ is a set of operations which may change the state of the component over time depending on the occurrence of an initiating event. A mandatory operation for sporadic components is the mechanism (real-time operations) containing all actions they provide.
$I_{\text{required}}^{\text{spor}}$ is a set of required interfaces which need a set of initiating events $E = ev_1,...,ev_n$ causing the activity of the sporadic component’s operation over time.

$I_{\text{provided}}^{\text{spor}}$ is a set of provided interfaces which may produce a set of outputs over time.

$M_{\text{spor}}$ is a set of mechanisms, whose mandatory attributes such as minimum arrival time or component deadline characterise the operation of a sporadic component.

$Contract_{\text{spor}}$ constrains the actions of the sporadic component. They are depicted within certain durations of time by marking the beginnings with start times and deadlines for component executions. Assuming that the minimum arrival time ($T_{\text{min}}$) is known, the component is constrained as follows:

$$S_{i,k+1} > S_{i,k} + T_{\text{min}}, \quad S_{i,k} < D < S_{i,k+1} \quad \text{where} \quad S_{i,k} \geq 0, \quad \text{and} \quad T_{\text{min}} \text{ is the (minimum) arrival time.}$$

3. Behaviour Specification

As mentioned above, the behaviour of the real-time component is described by provided and required behaviour typically captured as communication on the component’s provided and required interfaces.

As shown in Fig. 2, the real-time component state machine consists of states, events and transitions [8].

**State** depicts an active component satisfies some condition, performs some activity, or waits for some event. When a state is entered as result of a transition, it becomes active. Any state may optionally have one each of entry, exit, and do activities.

**Event** dispatch an event to its destination active component. An event expression is defined by transition rules which determine the pre-conditions and consequences of state changes. Each event carries an identifier of the active component that will receive the event.

**Transition** is a relation between two states indicating that a component in the first state will perform certain specified actions, and enter the second state when a specified event occurs and the specified conditions are satisfied. A transition rule is a Boolean equation, composed of event variables. When such an expression is satisfied, a transition from one state to another is enabled and system reconfiguration is to be performed.

**Action and Activity** are atomic computations that change the states of active components. They are considered as the fundamental units of behaviour specification.

![Real-time component state machine](image)

3.1. Component Sequence Diagram

A real-time component sequence diagrams (see Fig.3) are used to specify interactions among components that exchange messages within an interaction arranged in a time sequence. A sequence diagram focuses on each basic component, and the messages they exchange to accomplish some desired behaviour. It is typically used to show components interaction in a single scenario, and makes it easy to see the order in which activities occur.

![A simple sequence diagram](image)
As shown in Fig. 3, the vertical axis of a sequence diagram represents time, whereas the horizontal axis represents components participating in the interaction. Sequence diagrams consist of the elements:

- **Com_n** representing components that may participate within interactions;
- **Lifeline** representing the existence of a component over a certain interval of time;
- **Activation** representing the time during which a component is performing an operation, and
- **Message** representing communication between components.

At each component box a vertical line is drawn, referred to as a lifeline. Activation depicts a period when the component is executing (e.g., an operation of the component is executed), for which the lifeline is generally shown as a double solid line. Labelled horizontal arrows represent messages. Only source and destination of messages are relevant. Time increases from the top to the bottom. A sequence diagram provides a means to sketch possible sequences of action a system can go through. Some additional constraints were added in the component in state machine. Constraints, such as guard and pre- or post-conditions, etc., may be used for states and possibly transitions[5] [7].

4. Conclusion

Extensions of component-based design with static and dynamic specification possibilities for developing embedded and real-time systems are presented. Static specifications are extended by defining a real-time task component, which deals with real-time contracts for modelling non-functional requirements. State machines and sequence diagrams are used to express the behaviour of systems or of parts of them. To describe non-functional requirements, state machine are extended by defining timing and probability constructs, and sequence diagrams are enriched by some constraints which are contained in the specification of structure and behaviour. We represent them as additional constraints, and attach them to certain locations on the lifelines of component in a sequence diagram. The extension of the static and dynamic specifications aims to provide a means that enable to express real-time constraints.

References

[1] Brand,T. and Giese, H. (2018) Towards software architecture runtime models for continuous adaptive monitoring. In: 21st MODELS Workshops 2018: Copenhagen. pp.72-77
[2] Bate, I. Hawkins, R. and McDermid,J.A. (2003) Contract-based Approach to Designing Safe Systems. In: Safety Critical Systems and Software, Canberra, ACT, Australia, pp. 25–36
[3] Crnkovic, I. (2005)Component-based Software Engineering for Embedded Systems. in: Proc. 27th Intl. Conf. on Software Engineering, Pittsburgh. pp. 712–713
[4] Garlan, D. (2003)Formal Modeling and Analysis of Software Architecture: Components, Connections, and Events. in: Proc. 3rd Intl. School on Formal Methods for the Design of Computer, Communication and Software Systems, Bertinoro. pp. 1–24
[5] Harel, D. and Thiagarajan, P.S. (2003) Message Sequence Charts. In: UML for Real: Design of Embedded Real-time Systems, L. Lavagno, G. Martin and B. Seli´c (eds.), pp. 77–105, Kluwer Academic Publishers
[6] Lu, S-R. (2009) Engineering of Safety-related and Embedded Real-time Systems, Dissertation, ISBN: 978-3-89863-241-6, GCA-Verlag, Waabs
[7] Lu, S-R, Halang,W.A. Zhang,L-C.(2005) A Component-Based UML Profile to Model Embedded Real-Time Systems Designed by the MDA Approach, RTCSA'05, Hong Kong, pp. 563 – 566.
[8] Lu, S-R, Schmidt, H. W. and Gumzej, R. A (2005)Component-based Approach to Specify Hazards in the Design of Safety-Critical Systems, Proc. 3rd IEEE Intl. Conf. on Industrial Informatics, Piscataway,ISBN 0-7803-9095-4.
[9] Stankovic, J.A.(1988)Misconceptions About Real-Time Computing. IEEE Computer, 21(10): 10–19.