Real-Time Requirements Engineering (RTRE) as Extended to Hard and Soft Real-Time Analysis

Ahmed Tarek¹, and Ahmed Farhan²

¹Engineering, Physical and Computer Sciences, Montgomery College, Rockville, Maryland, United States of America
²College of Arts and Sciences, University of Pennsylvania, Philadelphia, Pennsylvania, United States of America

Abstract—Real-Time Systems (RTSs) have more stringent temporal requirements as compared to classical software. Requirements analysis for RTS is more complex compared to traditional software. Flowcharts as purely sequential tools are not sufficient for concurrent real-time systems. Besides, the requirements analysis leading to real-time software design varies from system to system contingent upon software needs, which has progressed into the relatively new discipline of Real-Time Requirements Engineering (RTRE). RTRE uses additional tools and techniques besides the traditional ones, such as, Process Activation Table, Decision Table, Decision Matrix, and so in the Structured Analysis leading to robust and secured software design. In this paper, we address some of the essential RTRE tools and techniques, and their significance to real-time analysis as extended to Hard and Soft RTS. Hard and soft RTS are not required to be embedded software systems. However, time still plays a crucial role in their operation. This extended approach yields with better design, coding and software reliability.


1. Introduction

There exists a wide variety of tools and techniques for requirement analysis of strictly sequential systems. For instance, Jackson System Development (JSD) [4] covers the complete technical development of a wide class of systems that are strictly sequential. Concurrency imposes additional restrictions to system development [1]. A real-time system needs to respond to stimuli that occur at different points in time. Therefore, it is necessary to organize its architecture so that, as soon as a stimulus is received, control is transferred to the correct handler. Hence, the system architecture must allow for fast transfer to the appropriate handler. This is impractical in sequential programs. As a result, real-time systems are designed as a set of concurrent, cooperating processes, with a real-time executive controlling the processes [3]. Based on the sensitivity to temporal requirements, there are two categories - soft and hard real-time systems. With real-time systems, program’s control flow is the most important part Ü as concurrency and control comes together and plays a vital role in the system operation. Therefore, there are two critical things to consider - the actions the program should take, and also the order of taking those actions. These two factors are related to Structure of the Code Ü how the code would look like. For classical software, simple, straight forward flowcharts are sufficient, where as for Real-time systems, programs are concurrent, and flowcharts are not enough as they are used to represent sequential control flow. This gave rise to the Real-Time Requirements Engineering (RTRE). The requirements engineering of real-time software deals with how requirements should be expressed, and also determines final formats of requirements specification.

In Section 2, specific terms and notations used in this paper are briefly deliberated. Section 3 explores RTRE. Section 4 considers the real time software design steps. Section 5 surveys the tools and techniques used for Real-Time Structured Analysis. Section 6 is the conclusion based on models of RTRE discussed in this paper, which also explores the future research avenues.

2. Terms and Notation

Following terms and notations are used all throughout this paper.

Soft Real-Time Systems: Time is of utmost importance but a little deviation from time bound does not lead to disastrous effects but only degrades the system performance. An example is an online banking system.

Hard Real-Time Systems: Time is the most critical factor, and a small deviation from time bound could be disastrous. An example is the real-time system used with a chemical processing plant.

Real-Time Requirements Engineering (RTRE): The requirements engineering for real-time software deals with how requirements for soft and hard real-time systems should be expressed. RTRE also determines final formats of requirements specification.

RTA: Real-Time Analysis.


CSPEC: Control Specification.

DFD: Data Flow Diagram.

CFD: Control Flow Diagram.
DD): Data Dictionary.
STD: State Transition Diagram.
PAT: Process Activation Table.
DT: Decision Table.
DM: Decision Matrix.
FSM: Finite State Machine.
FSA: Finite State Automata.
AID: Architecture Interconnect Diagram.
IP: Input Processing.
OP: Output Processing.
UIP: User Interface Processing.
MSTP: Maintenance and Self-Testing Processing.
CD: Context Diagram.
CCD: Control Context Diagram.

3. Real Time Requirements Engineering Explored

For real-time systems, program’s control flow is the most important part, as concurrency and control come together and plays a vital role in system operation. This incorporates but is not limited to the the following:

1) Actions of the Real-Time Software: The actions that the real-time program is supposed to take, when certain event occurs. For example, shut down the chemical processing plant in times of an emergency.

2) The Order of Taking Actions: The order in which actions are required to be performed is of paramount importance for real-time concurrent systems. This is in order to avoid any mishap. For instance, boiler needs to be shut down before stopping the plant operation. User information is required to be saved before logging out of an online account.

All these determine the structure of the code, which is how the code is to be designed, and how the code would look like. Therefore, for classical software, simple, straight forward flowcharts are sufficient. With real-time software operating in concurrent mode, flowcharts are not sufficient, as flowcharts are used to represent Sequential Control Flow. This has advanced the Real-Time Requirements Engineering (RTRE) for real-time system design and operation. The requirements engineering of real-time software (RTS) deals with how requirements should be expressed for designing concurrent, real-time systems. RTRE also determines the final formats of the requirements specification.

RTSs are almost always related to the surrounding environment with which the system interacts. As a result, for RTS, the requirements are grouped into categories as follows:

1) Input requirements relating to input signals or data.
2) Output requirements relating to output signals or data.
3) Joint I/O requirements. These are related to input and output data simultaneously.
4) Processing requirements relating to internal function of software.

In addition to the above functional requirements that relate to the functionality of real-time software, there is a group of other requirements relating to certain attributes of software such as, performance, safety, reliability, security, etc. The discussion in this research paper on RTRE will highlight and illustrate these diverse issues that come naturally with real-time system design and construction.

3.1 Advantages Rendered Through RTRE

Real-Time Requirements Engineering offers added tools and techniques for robust analysis that ensures safety, security and system reliability. Following are the advantages rendered through RTRE.

Advantages of RTRE:

For Hard and Soft Real-time Systems, with RTRE:

1) Flow control analysis for software becomes easier.
2) Yields better software architecture.
3) Less chance of errors in Real-time Software Design.
4) RTRE provides with a robust analysis leading into robust and reliable software design.
5) As the design is robust, there is less chance of changing the software design or requirements later resulting in reduced cost of development.
6) Eventually, this robust process results in better real-time software testing strategies.

From Software Analysis to Software Testing with RTS, it’s a complete process known as the Structured Approach.

4. Real Time Software Design Steps

The Hatley & Pirbhai approach as articulated in [1] is one of the best approaches to real-time software design. The modified Hatley and Pirbhai approach starts designing the software with the Context Diagram, and follows through seven design steps.

2) Perform Control Flow Decomposition through Control Flow Diagrams (CFDs).
3) Develop Process Specifications (PSPECs) in Structured English.
4) Develop Control Specifications (CSPECs) using State Transition Diagrams (STDs).
5) Develop Response Time Specifications (RTSs) for timing thresholds.
6) Develop Data Dictionary (DD).
7) Produce Architecture Interconnect Diagrams (AIDs).

The key to developing real-time software is understanding the relationship of software with the surrounding environment, with which, the software interacts. For instance, for Airline Ticket Reservation software, the real-time software interacts with the customers, who are trying to buy the
tickets. The principal ways of expressing these interactions are:

1) A Physical Diagram (PD), representing all physical entities, including a computer system or systems, if applicable. This leads to Physical Diagram.
2) A Context Diagram (CD) is a refinement of a Physical Diagram that represents the Real-Time Software as a single bubble (circle), and all external devices as rectangles. There are arcs connecting the circle, and the rectangles representing connections, data flow, and also the directions of data flow.

The key elements in the design with this approach are based on the following:
1) Deriving DFDs and CFDs using the template of the architecture.
2) Relies on the main characteristics of the system as captured by DFDs and CFDs.
3) But the design refines DFDs and CFDs into a physical representation.

The real-time software design template is based on 5 components.
1) Functional and Control Processing (FCP), which constitutes the Central Box of the design template. This part is based on Process Model (PM) and Control Model (CM) of the real-time software.
2) Input Processing (IP), constituting the left of central box.
3) Output Processing (OP), which makes up the right of central box.
4) User Interface Processing (UIP), shown as the top of central box.
5) Maintenance and Self-Testing Processing (MSTP), represented as the bottom of central box.

Following figure is representative of these five different components and their relative positioning in the design template.

Whenever it comes to the actual system implementation with the real-time embedded software, following notations and symbols play major roles. Following figure is representative of different notations used during this phase. Following

Fig. 1: Software Design Template.

represents the ideal scenario in real-time embedded system design and development.
1) Software Engineers meet with the Client to collect and gather the System Requirements(SRs).
2) Based on the collected SRs, Real-Time Requirement Analysis (RTRA) takes place.
3) Once RTRA is complete. Real-time Software Design Engineers pursue the System Design.
4) Once the Real-Time Software Design (RTSD) becomes available, both the Hardware Development (HD), and the Software Development (SD) phases progress in parallel.

Following are the typical sub-phases during HD phase.

a) Hardware Requirements Analysis (HRA)
b) Preliminary Hardware Design (PHD)
c) Detailed Hardware Design (DHD)
d) Hardware Fabrication (HF)
e) Hardware Testing (HT)

Following are the standard sub-phases during real-time embedded software development.

a) Real-Time Requirements Analysis (RTRA)
b) Preliminary Software Design (PSD)
c) Detailed Software Design (DSD)
d) Real-Time Software Coding and Unit Testing (CUT)
e) Real-Time Software Integration Testing (IT)
5) Once both the hardware and software become available, System Integration (SI) is performed.
6) Once SI is complete, System Integration Testing (SIT) and System Testing (ST) are done to ensure safety, security and reliability.

5. Tools and Techniques for Real-Time Structured Analysis

Whereas the Classical Software Engineering (CSE) typically uses Data Flow Diagrams (DFDs), the Real-Time Requirements Engineering (RTRE) includes, but are not limited to the following tools and techniques. This is due to the nature and criticality of the real-time system design, implementation and physical operation.

1) Data Flow Diagrams (DFDs)
2) Control Flow Diagrams (CFDs)
3) Decision Tables (DTs)
4) Decision Matrices (DMs)
5) Process Activation Tables (PATs)
6) State Transition Diagrams (STDs)
7) State Transition Tables (STTs)

This paper mostly focuses on the current tools and techniques pertaining to RTSE. Using Structured Approach for RTA yields with added benefits. Some examples of real-time systems include Vending Machine, On-line Banking websites, etc. These are soft real-time systems. With hard real-time systems, timing factor is crucial, such as the chemical processing plants.

The Structured Analysis Model (SAM) uses the following core concept: Following constitutes the basic components of SAM.

1) Data Dictionary (DD): DD represents core of the model. DD is a repository containing a description of all data objects.
2) Entity Relationship Diagram (ERD): ERD depicts relationships between data objects formulating data attributes.
   a) DFD describes how data is transformed as it moves along the system.
   b) DFD depicts the functions for transforming the data flow.
4) Process Specification (PSPEC): The description of each transform function for data flow is contained in PSPEC.
5) State Transition Diagram (STD): STD describes how the system responses to external events.

5.1 Real-Time Analysis (RTA)

Following are important segments of Real-Time Analysis (RTA).

2) State Transition Analysis (STA): Performed through Control Specification (CSPEC).
3) Data Dictionary (DD): A repository of data items useful for DFA and STA.

Following implies how the Analysis Model (AM) maps into a hierarchy referred to as the Design Model (DM).

1) Component Level Design (CLD): CLD constitutes the Highest Level (HL) in the entire hierarchy. CLD is derived from Process Specification (PSPEC), State Transition Diagram (STD), and Control Specification (CSPEC).
2) Interface Design (ID): ID is derived from State Transition Diagram (STD), and Control Specification (CSPEC), as well as the Data Flow Diagram (DFD).
3) Architectural Design (AD): AD is the next lower level in the hierarchy. It uses the Data Flow Diagram (DFD).
4) Data Design (DD): DD is the Lowest Level (LL) in the entire hierarchy. It is derived from Data Dictionary (DD), and the Entity-Relationship Diagram (ERD).

5.2 On Tools and Techniques for RTRE

5.2.1 Data Flow Diagram (DFD)

Shows how data is transformed as it moves along the real-time system. DFD is a graphical aid for RTA. DFD uses following symbols. DFDs start with the Context Diagram (CD), which represents the entire system as a big bubble and proceeds through the hierarchical decomposition. Following illustrates the principle of hierarchical decomposition of designs.
5.3 Illustration Of Data Decomposition:

To better illustrate the process of data decomposition, consider the example of a vehicle Cruise Control System (CCS). Following represents the Context Diagram (CD) of the CCS. As evident from the figure, the CCS interacts with its surrounding environment, which in this case, consist of, throttle, cruise buttons, brake, engine, speed sensor, and the gear assembly.

Next for the control purposes, the Cruise Controller Process is decomposed into lower level functions as shown in the following. As evident from the CCP decomposition, there are three main functions of the cruise controller process, which are:

1) Select Speed: Select a particular speed using the cruise controller button.
2) Maintain Speed: Maintain the pre-set vehicle speed selected through the cruise controller button.
3) Maintain Acceleration: Maintain acceleration to preserve the vehicle speed to the pre-set level by controlling throttle (fuel injection).

Since all three of these functionalities are related to control, the entire cruise controller in which, these three control functions are nested is shown with a broken bubble, which represents a Control Context Diagram (CCD). The PSPEC for the above three control functions follow: PSPEC represents the textual description of control logic related to each one of the three control functionalities. The change of state in the control application is best represented by a State Transition Diagram (STD). Following represents STD of the Cruise Controller. In context to STDs, Finite State Machines (FSMs) play a vital role. Following subsection discusses FSM in relation to real-time systems.

5.4 Finite State Machine (FSM) in RTRE

Finite State Machine (FSM) plays a major role in RTRE. Real-time systems have the property of past and present events. Also, both external and internal events can change
real-time behavior. Real-time systems respond to changes in their input, and also to events that occur over time with system appearing as if several different processors from one instance of time to another. The process model alone fails to express this type of action. Therefore, FSM combined with the process model is suited to modeling the change in state behavior due to internal or external events.

Continuous Machines (CMs) or Analog Machines (AMs) are capable of processing continuous-valued inputs, outputs, and internal elements. CMs are also able to receive and produce discrete-signal values. PSPECs represent continuous machines in the requirements model.

On the contrary, FSMs can only process discrete signals. Control specifications (CSPECs) are represented by FSMs. There are two categories of FSMs.

1) Combinational Machine (CM) Û Combinational Machines have no memory. As a result, CM can not hold or represent the past states. It is capable of representing only the present states.

2) Sequential Machine (SM) Û SM has Memory. As a result, SM is capable of representing both the past and the present states. SM requires a wide a variety of tools to show the state transitions. These include, but are not limited to STD, DT, DM, PAT, etc.

5.5 Decision Table (DT) and Decision Matrix (DM) in RTRE

DTs and DMs are often used in RTRE for structured analysis. An example follows:

The Decision Table (DT) for organizational employees helps in making decisions regarding salaries. DTs are two dimensional and composed of rows and columns. Each of the columns defines the conditions and actions of the rules. Decision tables represent a list of causes (conditions) and effects (actions) in a matrix, where each column represents a unique rule. The purpose is to structure the logic for a specific specification feature. It is possible to add new rows to a decision table and fill in its cells to create new rules. When the rules are executed, if the conditions are met, the actions in that column are performed. One can also add preconditions that apply to all the rules.

5.6 Process Activation Table (PAT) in RTRE

Decision Tables (DTs) that are used to activate processes are called Process Activation Tables (PATs). Following represents a CSPEC containing both PAT and DT.

5.7 STDs & DTs In RTRE

Following figure illustrates CFD/DFD (left diagram) and transitions among states due to the event triggering (right diagram).

Fig. 10: DT for Employees.

Fig. 11: CSPEC with PAT and DT.

Fig. 12: STD in RTRE.

Fig. 13 is representative of CFD/DFD (left diagram) and the decisions taken due to different combinations of the occurring events (right diagram).

Sometimes DFD and CFD are combined in CSPEC to represent CSPEC as the STD. Fig. 14 is representative of this scenario.
6. Conclusion

State Transition Diagram is only suited for a limited number of states. Alternative representations used with large number of states are State Transition Matrix (STM) and State Transition Table (STT). Control Specifications (CSPECs) consists of FSMs, which map control inputs to control outputs. CSPEC combines both Combinational and Sequential Machines. Block diagram in Fig. 15 is representative of this scenario. A general real-time system model involves associating processes with sensors for data input, and actuators for data output, which are sent as control signals. Real-time System architecture is usually designed as a number of concurrent processes. Real-time system specification involves System Requirement (SR), representing the Logical Configuration of software, and System Architecture (SA), representing the Physical Configuration of software. Both SR and SA are required to be developed in parallel.

After incorporating different components in the Software Design Template (SDT) for Cruise Controller, following Fig. 16 is the complete software architecture.

System functionalities are either partitioned to hardware or to software. Design decisions are made based on System Requirements. Hardware implementation delivers better timing performance compared to the software counter-part. However, functional implementation with Hardware incurs higher costs, potentially longer development time, and less flexibility for accommodating any future changes.

Due to functional requirements, sometimes RTS may not be implemented using the Object-Oriented Approach. Real-Time Requirements Engineering (RTRE) is wide. There are still unexplored avenues in this engineering domain. We tried to represent a few of the useful tools and techniques from the RTRE paradigm in connection to modern real-time systems. In future, we plan to incorporate RTRE, and the Structured Approach discussed in this paper for a realistic Patient Monitoring System (PMS) at hospitals, and also to an Online Registration Control System (ORCS) for a self-sustained small academic institution.

References