Real-Time Analysis Process Patterns

Naeem Esfahani[1], Seyed-Hassan Mirian-Hosseinabadi [2], Kamyar Rafati[3]

[1] Computer Engineering Department, Sharif University of Technology,
Tehran, Iran
esfahani@ce.sharif.edu

[2] Computer Engineering Department, Sharif University of Technology,
Tehran, Iran
hmirian@sina.sharif.edu

[3] Computer Engineering Department, Sharif University of Technology,
Tehran, Iran
rafati@ce.sharif.edu

Abstract

The influence of Object Oriented Modeling is observable in various areas of software engineering. Embedded and Real-time system domains are not exceptions; several object oriented methodologies have been proposed in these domains. These methodologies have many concepts in common, but the diversity in presentation has concealed their similarities. In this paper, these commonalities in requirement modeling and analysis are captured as process patterns. We also present a generic workflow which covers the requirement modeling and analysis phases in real-time methodologies. For this purpose, several real-time methodologies such as Octopus, STARSS, DESS, Comet and RT-UML have been studied and analyzed focusing on requirement modeling and analysis. As a result, their common parts are extracted as three process patterns. At the end, the occurrence of these patterns in each methodology has been elaborated. It is concluded that the effect of real-time considerations in these phases is restricted to modeling, problem domain partitioning and emphasis on state-driven analysis.

Keywords

Real-time system, Analysis, Requirement Modeling, Process Pattern, Embedded system

1. Introduction

Scheduling, robustness and safety are the main properties of real-time systems. Not paying enough attention to any of these parameters may turn to a disaster. These systems often remain operational for a long period of time and may not be restarted frequently like desktop applications [1].

Usually real-time software is deployed as the heart of a larger system. Users do not deal with the embedded system like a desktop system; e.g. these systems have no keyboard and monitor. They are usually considered as electromechanical systems which provide a set of services for the user. The real-time software monitors and controls the hardware using sensors to acquire information about the system and actuators to perform actions in the outside world.

There are further more problems in real-time software development; e.g. we cannot simulate the whole system execution because hardware facilities such as timers and A/Ds are so fast to be emulated over a desktop computer during development. These dependencies between the development process and target platform take a lot of effort and time. The development team must test and verify the software over the final system.

Software development environments provide little facilities for real-time software development. This makes it obvious that these systems become more bug-prone. So a more thoughtful methodology is essential for managing the production of these systems.

One of the first and most important works performed in late 70's by introduction of MASCOT notation [2]. Later on a design method with a similar name was proposed [3]. Structured analysis and design in real-time development was introduced by RTSAD [4], Ward-Mellor [5] and DARTS [6]. Author of DARTS introduced another methodology named CODARTS [7]; this new methodology integrated proposed concepts from well known methodologies [8], [9,] [10].

Octopus methodology [11], [12] –an object oriented methodology– is based on Jakobson's use case and Rumbaugh's static modeling. It used UML-liked notations. Room methodology [13] is a real-time software design method that depends on a tool named ObjectTime.

After a while, Douglass described how to use UML in real-time software production in his books. In his first book [14], he illustrated how to use UML in modeling real-time systems and in the second one [15], he surveyed the real-time concepts. One of the recent real-time methodologies was COMET [16]. Introduced in 2000, it mainly focused on analysis and design phases. In another research [17], the analysis phase of some real-time methodologies has been studied.

DESS methodology [18] is another methodology for real-time system development. It uses object oriented in conjunction with component oriented modeling power. This methodology has inherited its concepts from RUP and V-Model. In 2006, STARSS [19] was introduced which is a methodology for fault tolerant real-time system development. It has a special emphasis on validation and verification.

So far, real time embedded methodologies have been reviewed. Ambler, in his two books, [20] and [21], has introduced process patterns. These patterns deal with more general concepts in software development, but less attention is paid to software development lifecycle. The main goal of process patterns is to review the overall lifecycle. In [22] and [23], some notations have been introduced for expressing process patterns.

Embedded real-time methodologies are not well-addressed in the context of process patterns. Trying to extract patterns from embedded real time methodologies, we observed that differences between these methodologies and general purpose software development methodologies are mostly in the analysis and design phases. The other parts of the development lifecycle are common between real-time and non real-time methodologies.

In this paper, we are going to present the results of our studies about embedded real-time methodologies. We captured the common concepts of these methodologies as a set of process patterns. In the next section, some well known real-time methodologies are reviewed. In the third section, the proposed process patterns are presented. We present the generic requirement modeling and analysis work flow in the fourth section. Finally, the paper is concluded in the fifth section.

2. Real-Time OO Methodologies

2.1. Octopus

This methodology was first proposed by Nokia Corporation in 1995. It was intended to bridge the gap between Real-Time systems and Object Oriented Technology.

Octopus contains two sequential phase (i) requirement specification and (ii) system architecture. In these phases, system is divided into subsystems and alternative solutions are specified. After these, parallel development of subsystems begins. Parallel development consists of analysis, design and implementation.

Software requirements are captured in requirement specification phase. Exact problem definition is prepared. Context diagram is used to show the structure of the problem environment. Functional and dynamic behavior of the system is specified by using use case diagram and use case sheets. Use case diagrams indicate system services and their relations which act as index for use case sheets.

In system architecture phase, system is partitioned as concurrent distributed subsystems which are more manageable. Partitioning is mainly based on work domain. A responsibility sheet is associated with each subsystem that states its duties. This causes a direct link between use cases and the subsystems; thus requirement traceability will be achieved. Functional model is a set of responsibility sheets. Dynamic model is formed from use case assignment diagram. In this phase subsystems are considered as black boxes.

Analysis is done for each subsystem; structure model (class diagram and class description table) is prepared. In analysis, class diagram has a pivoting role because it contains common concepts of functional and dynamic model. Functional model is a set of operation sheets. An operation sheet demonstrates operations of a subsystem in a structured manner using class diagram terms. Dynamic model contains event list, event group diagram, event sheets, state diagram, priority table and possibly composite event list.

Both the functional and dynamic models treat the subsystem as a black box and make references to the classes in the structural model.

2.2. DESS

This methodology was proposed as a tool for using object oriented and component oriented programming in Real-Time Embedded systems. This method contains three concurrent v-shaped workflows: (i) Realization, (ii) Validation-and-Verification and (iii) Requirement Engineering

Main system design and implementation tasks are done in Realization workflow. In this workflow, system and software requirements are captured. Then the software is analyzed and designed. When implementation is completed, software and system integration processes start and finally the system is deployed in the user environment.

Identification of bugs and weaknesses is done in validation and verification workflow. This workflow contains tasks such as testing, review, model checking, simulation, formal proof of correctness and symbolic execution. This workflow acts concurrent with Realization workflow and validates it.

The last V shaped workflow belongs to requirement engineering. Its main purpose is making an agreement with costumers. This workflow consists of planning, requirement engineering and management in traceability, change and report.

2.3. STARSS

This methodology was introduced for Real-Time control oriented in 2006. It has a bottom up approach to system analysis and design. It starts analyzing and designing the outer parts of the system (Sensors and Actuators) and continues to the inner parts till the central controller. This approach causes an onion like layering such that inner layers control the outer layers. Each layer will be produced at least in one iteration.

Before starting with layers, preparation phase is done; in this phase, system entities and their relations are identified. System communication devices are also identified in this phase.

After preparation, construction of specification and communication of layers starts. For each layer, entity sets and compositional structures are constructed, plugs of each controller are grouped, behavior of controller are modeled, event list and object dependencies are identified, any kind of dependency between components and object is captured. Finally state diagrams are reviewed and specifications are defined.

2.4. Comet

Comet was introduced by Gomaa in 2000 [16]. In this methodology he tried to add object oriented flavor to his previous methodologies [6].

In comet requirement modeling is done in requirement elicitation stage. The functional requirements of the system are described using actors and use cases. In use case description, sequence of actor(s) interaction with the system is revealed. In Embedded-Real time systems actors include input-output devices and timers.

In the analysis model we suppose to become familiar with the system. This stage consists of two steps; in the first step we build static model of the system and structure of objects. Building a dynamic view of the system is the subject of the second step.

By static modeling the classes (Attributes and Operations) and their structural relations are modeled using class diagram. The first emerging classes are the physical and entity classes. The class diagram is used as a context diagram to help understanding the interface of the system to environment.

In dynamic modeling, the dynamic model is completed; and the related objects are clarified. The dynamic analysis is two-fold; inter-object behavior is modeled using collaboration diagram and intra-object behavior, which is a state-driven behavior of the object, is modeled using the state transition diagram. Incoming and outgoing messages of state-driven objects in the collaboration diagram match the input and output events in the state transition diagram. This close relation between two models makes the task of their construction iterative in nature.

2.5. RT-UML

This methodology was introduced in 1999 by Douglas. In requirement modeling, the external classes and their relationship with the system are realized using the context diagram. In this diagram, the message passing between system and its environment is modeled.

After identifying the environment and the outside world, structure definition will be started. First, essential objects and classes and their relationship and hierarchies should be discovered. The starting point for this task is the context diagram which was resulted in the previous phase. The resulting model will be enriched in dynamic modeling.

There is an iterative cycle for finding classes. The first half (Identify Objects, Identify Object Associations, Group Objects into Classes, Identify and Classify Class Relationships) of it is carried out in the structure definition phase and the other half is done in the behavior definition phase.

The object behavior, relations and attributes completes knowledge about object responsibility; these responsibilities will be embedded in the operations. The second part of the cycle (Group Classes into Domains, Validate Classes and Objects) can be performed knowing object-class responsibilities we can perform.

If the system has state-based behavior, it means that the system has memory of previous services and how they affect the next services, and then it must be modeled using state diagram.

3. Process Patterns

Comparing existing methodologies and creating a special purpose methodology requires an overall knowledge. Abstracting existing methodologies helps in gathering required information for achieving this goal. Abstraction also highlights the most important stages in the development process.

Extracting recurring patterns in methodologies (process patterns) is the key tool in methodology abstraction. These patterns show common features of methodologies. They also show essential activities which should be performed in every methodology, regardless of the specific heuristics and approaches. Furthermore, when we are composing a new methodology or configuring an existing one, these patterns help us to select appropriate elements and consider process alternatives.

In this section, we introduce three process patterns for requirement modeling and analysis of real-time systems. For each pattern, the following information has been gathered:

- **Base Methodologies:** The methodologies from which this pattern is extracted.
- **Purpose:** The outline of the concept which is realized by this pattern.
- **Pattern Phase:** The phase or stage in the development process that this pattern contributes to.
- **Inputs and Outputs:** The input and output artifacts of the pattern.
- **Term Definitions:** The glossary of ambiguous terms.

3.1. Requirement Elicitation

This pattern (Fig 1, Table 1) identifies software requirements. It captures software functional model using use case diagram and use case descriptions. For complex use cases, Alternative flows are introduced. Main flow and alternative ones form events scenarios in the system. The system is considered as a black box.

Table 1. Requirement elicitation process pattern summary

Base	DESS, Octopus, RT-UML, COMET	
Purpose	Elicit and model software requirements	
Phase	Initiation	
Inputs	Internal	System requirements
		System design
		HW/SW allocation
	External	Standards
		Organization policies
		Customer inputs
Outputs	Software Requirement model	
Terms	Use case	Event sequence that starts from an actor (shows actor's interaction with system). It is possible to have multiple actors engaged in a use case who fire various events in different times.

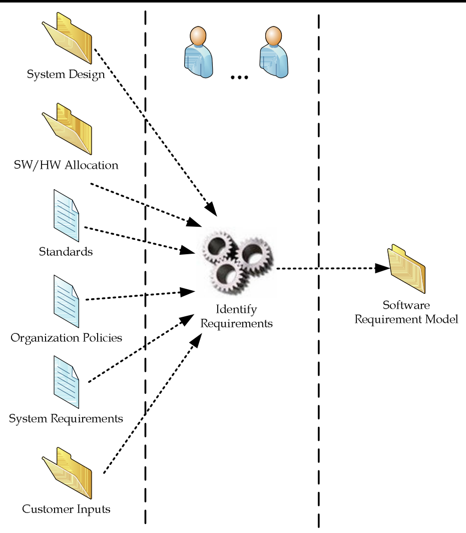

Figure. 1. Requirement elicitation process pattern diagram

3.2. Static Modeling

In an iterative fashion the scenarios of the use cases are reviewed and enriched. Required objects and relationships for use case realization are captured. Similar object are grouped into the classes and class relationships are extracted. According to their usage domain, the classes are categorized (Fig 2, Table 2). In this process following activities are carried out:

- **Identify objects and their relationships:** object(s) that are required for use case realization is identified. Their relationships mainly are based on message passing.
- **Identify classes and their relationships:** similar objects are recognized and their analogous attributes and operation are classified using classes. Corresponding object relationships are captured as class relationships.
- **Class categorization:** according to their functionality, classes are grouped.

Table 2. Static modeling process pattern summary

Base	STARSS, Octopus, RT-UML, COMET	
Purpose	Prepare structural model required for use case realization	
Phase	Construction	
Inputs	Internal	Software requirement model
	External	Standards
		Organization policies
		Problem domain Information
Outputs	Structural model	
Terms	Structural Model	The set of classes, objects, their attributes, operations and relationships.

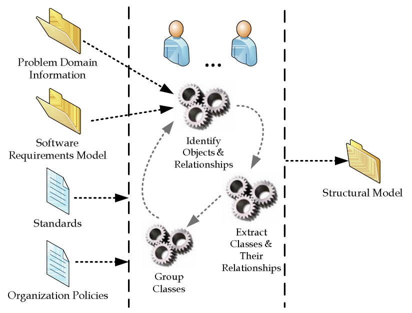
Problem Domain Information
Software Requirements Model
Standards
Organization Policies
...
Identify Objects & Relationships
Extract Classes & Their Relationships
Group Classes
Structural Model

Figure. 2. Static modeling process pattern diagram

3.3. Dynamic Modeling

Focusing on the main scenarios, object communications are captured; the event sequences and state transitions (in different levels) are identified. These sequences and transitions are enriched using alternative flows. It is possible to add new objects and classes to capture all behaviors (Fig 3, Table 3). This pattern has two activities:

- **Identify event sequences:** communications among object which are captured in static modeling, event sequences and message passing among them are modeled.
- **Identify state transitions:** as a response to the events in the system, there might be some state changes in different levels. These transitions are modeled concurrently with event sequences.

Table 3. Static modeling process pattern summary

Base	STARSS, Octopus, RT-UML, COMET	
Purpose	Prepare behavioral model that describe communications among structural elements and operation sequences in order to realize use cases.	
Phase	Construction	
Inputs	Internal	Software requirement model
		Structural model
	External	Standards
		Organization policies
Outputs	Behavioral model	
Terms	Behavioral Model	Expresses intra- and inter-object behaviors required for use case realization.
	Scenario	Describes a sequence of events, state transitions and operations.

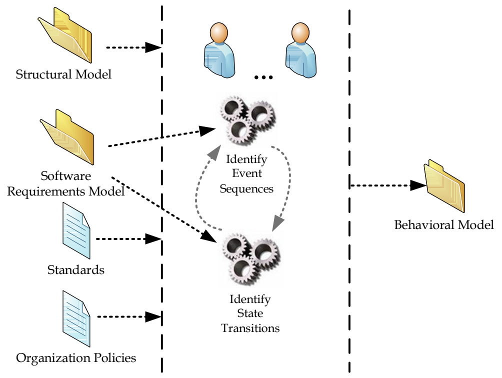

Figure. 3. Dynamic modeling process pattern diagram

3.4. Process patterns in the methodologies

In Appendix A, we have shown how the proposed process patterns related to the studied methodologies. How these abstract patterns live in the concrete methodologies is understood.

4. Generic Workflow

To complete our patterns, the order and relationship among them should be illustrated. This arrangement could be accomplished by a closer look at the inputs and outputs of the patterns.

In this section, we present a generic workflow for requirement and problem domain modeling. This generic workflow is extracted from the studied methodologies and the indicated order is the same in all of them. Also other arrangements are possible [17].

As depicted in Figure 4, requirements are modeled in the requirement modeling phase. Next, the models are analyzed; first the static model of the domain is built and the result is further completed by adding behavior in dynamic modeling. Static and dynamic modeling are performed iteratively; The result is improved gradually.

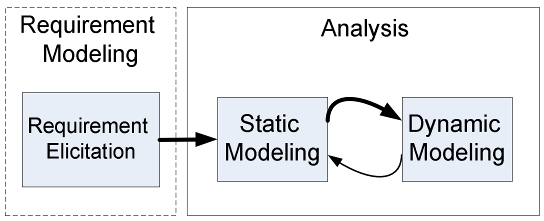

Figure. 4. Generic workflow

5. Conclusion

The focus of this paper is on requirement modeling and analysis of embedded real time systems. The existing real time embedded methodologies have employed diverse mechanisms for this purpose, but they have followed analogous policies. We gathered and presented their similarities as three process patterns.

In requirement elicitation, user expectations of the system are captured as comprehensible models. By performing static modeling, the structure of the system is captured as a set of collaborating classes. The model presented in static modeling is enriched by insertion of behavior when performing dynamic modeling.

The general purpose object oriented methodologies have concepts in common with real-time methodologies studied, but significant differences exist in their point of view to the problem domain; e.g. modeling non functional requirements, especially time, is more crucial. Interactive nature of real-time systems leads us to a special kind of system decomposition: system is viewed as several enveloping layers. In these systems, controller plays a critical role; so, state driven analysis becomes important for supporting history dependent decision making.

Currently we are continuing our research to elaborate the next phases, particularly the design phase. Preliminary results show that real-time nature of these methodologies is even more significant in design and implementation. As a future work, we are going to introduce a generic lifecycle for real-time object oriented methodologies.

Appendix A. Process pattern occurrence matrix

Table 4 summarizes the presented information in the second section to show how the presented methodologies are covered by introduced process patterns.

Acknowledgment

The authors gratefully acknowledge Dr. R. Ramsin from Sharif University of Technology for his suggestions and Mr. A. Moghimi from University of Tehran for verifying some parts of the paper.

References

[1] Kopetz, H. Limmerick, Software engineering for real-time: a roadmap. Ireland : IEEE, 2000. IEEE Software Engineering Conference. pp. 201-211.

[2] Simpson, H. and Jackson, K. Process Synchronization in MASCOT. 1979, The Computer Journal 17 no. 4.

[3] Simpson, H. The MASCOT Method. Addison-Wesley, 1986, IEE/BCS Software Engineering Journal 1, no. 3, pp. 103-120.

[4] Hately, D. and Pirbhai, I. Strategies for Real Time System Specification. New York : Dorset House, 1988.

[5] Ward, P. and Mellor, S. Structured Development for Real-Time Systems Vols 1, 2 and 3. New York : Yourdan Press, 1985.

[6] Gomaa, H. 1984, A software Design Method for Real-time Systems. Communications, pp. 938-949.

[7] Gomaa, H. Software Design Method for Concurrent and Real-time Systems. s.l. : Addison-Wesley, 1993.

[8] Jackson, M. System Development. Englewood Cliffs, N.J. : Prentice Hall, 1983.

[9] Parnas, D., Clements, P. and Weiss, D. The Modular Structure of Complex Systems. Orlando, Fla., 1984. 7th IEEE International Conference on Software Engineering.

[10] Booch, G. Object Oriented Design with Applications. s.l. : Addison Wesley, 1991.

[11] Awad, M., Kuusela, J. and Ziegler, J. Object-oriented Technology for Real-time Systems: A Practical Approach

Using OMT and Fusion. Englewood Cliffs, NJ : Prentice Hall, 1996.

[12] L., Ziegler, Awad, M. and Kuusela, J. Applying object-oriented technology in real-time systems with the OCTOPUS method. IEEE, 1995. First IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'95). p. 306.

[13] Selic, B., Gullekson, G. and Ward, P. Real-Time Object-Oriented Modeling. New York : Wiley, 1994.

[14] Douglass, B. P. Real-time UML 2nd ed. s.l. : Addison-Wesley, 1999b.

[15] Douglass, B. P. Doing Hard Time: UML, Objects, Frameworks, and Patterns in Real-time Software Development. s.l. : Addison-Wesley, 1999a.

[16] Gomaa, H. Designing Concurrent, Distributed, and Real-time Applications with UML. New York : Addison-Wesley, 2000.

[17] Kimour, Mohamed T. and Meslati, Djamel. Deriving objects from use cases in real-time embedded systems. 2005, Information & Software Technology 47(8), pp. 533-541.

[18] Baelen, Stefan Van, Gorinsek, Joris and Wills, Andrew. The DESS Methodology. Version 01 - Public, Deliverable D.1, 2001.

[19] Kan, Pauline. The STARSS Methodology, Department of Computer Science, King's College London, Technical Report TR-06-03, 2006.

[20] Ambler, S. W. Process Patterns: Building Large-Scale Systems Using Object Technology: Cambridge University Press, 1998.

[21] Ambler, S. W. More Process Patterns: Delivering Large-Scale Systems Using Object Technology. Cambridge University Press, 1999.

[22] Gnatz, M., Marschall, F., Popp, G., Rausch, A., and Schwerin, W. 2001. Towards a Living Software Development Process Based on Process Patterns. 8th European Workshop on Software Process Technology (June 19 - 21, 2001).

[23] Störrle, H. 2001. Describing Process Patterns with UML. 8th European Workshop on Software Process Technology (June 19 - 21, 2001).

Table 4. The process pattern occurrence in the studied methodologies

Method	Requirement Modeling	Static Modeling	Dynamic Modeling
Octopus	In the requirement specification phase, after problem definition, use case diagrams and use case sheets are produced. More complicated use cases will have transaction scenarios.	In the iterations of the subsystem analysis phase, the class diagram and class specifications are extracted.	After identification of the events, the communication scenarios are determined. The sequence of the events and their results are clarified. For state-driven objects, states are recognized and state diagram is drawn.
DESS	The first step in the realization process is user requirement management. Requirements are stated as business use cases.	Has not defined in this method but it has indicated the structure diagram as output of analysis phase. So it should be prepared somehow.	Again has not been mentioned in this process but the dynamic model is input of the design phase so it must be produced.
STARSS	Pay no attention to this step. It assumes the tasks start with analysis, where we have requirements.	At the beginning of the iterations for generating one level, static model is built. The objects of that level are identified; they will be classified as classes. The relationships are also modeled.	At each level, behavioral model is constructed using state transition diagram. Non state driven behavior is modeled by extracting events and their dependencies. In this way all the valid sequences will be identified.
RT-UML	The system context diagram is created. The messages and events between the system and environment are characterized. The use case diagram is produced. The scenario for each use case is extracted.	After understanding the outside world and system's relationship with the world, we step in the system and identify objects, classes and their relationships. We also use scenarios and use cases in objects identification process.	Using scenarios, the intra object relations are modeled. Object has a state diagram which determines their behavior. The way objects interact determines what portion of the state diagram will be used in each scenario.
COMET	The system is modeled as a black box. Relationships among the system and the actors are stated in a narrative way. The relationships are defined in use case diagrams.	The physical and the entity classes are more important. After identifying classes we classify them in the groups and subsystems.	The inter- and intra-object behavior is modeled iteratively. Objects' internals are modeled using state diagrams (if object has state driven behavior). Inter-objects behavior is modeled using collaboration diagrams. These two diagrams must be compatible.