Requirements Eng (2005) 10: 22–33
DOI 10.1007/s00766-004-0192-6

ORIGINAL ARTICLE

Hossein Saiedian · Prabha Kumarakulasingam
Muhammad Anan

Scenario-based requirements analysis techniques for real-time software systems: a comparative evaluation

Received: 15 February 2003 / Accepted: 13 January 2004 / Published online: 14 April 2004

Abstract One of the most critical phases of software engineering is requirements elicitation and analysis. Success in a software project is influenced by the quality of requirements and their associated analysis since their outputs contribute to higher level design and verification decisions. Real-time software systems are event driven and contain temporal and resource limitation constraints. Natural-language-based specification and analysis of such systems are then limited to identifying functional and non-functional elements only. In order to design an architecture, or to be able to test and verify these systems, a comprehensive understanding of dependencies, concurrency, response times, and resource usage are necessary. Scenario-based analysis techniques provide a way to decompose requirements to understand the said attributes of real-time systems. However they are in themselves inadequate for providing support for all real-time attributes. This paper discusses and evaluates the suitability of certain scenario-based models in a real-time software environment and then proposes an approach, called *timed automata*, that constructs a formalised view of scenarios that generate timed specifications. This approach represents the operational view of scenarios with the support of a formal representation that is needed for real-time systems. Our results indicate that models with notations and semantic support for representing temporal and resource usage of scenario provide a better analysis domain.

H. Saiedian is a member of the Information & Telecommunication Technology Center at the University of Kansas. His research was partially supported by a grant from the National Science Foundation (NSF).

H. Saiedian (✉) · P. Kumarakulasingam
Electrical Engineering & Computer Science,
University of Kansas,
Lawrence, KS 66045, USA
E-mail: saiedian@eecs.ku.edu

M. Anan
Sprint Corporation, Overland Park,
KS 66211, USA

Keywords Real-time · Requirements engineering · Scenario-based · Time-automata

1 Introduction

Requirements analysis paves the way for high-level design, generation of test cases for verification, and supports early architecture reviews. As a result of such analysis, analysts and architects gain a deeper and thorough understanding of the system to be engineered. A popular method of capturing requirements is using a case-based approach [1]. In this method, requirements are defined from the perspective of all actors (users as well as hardware and software) at a hierarchical level. However, these hierarchical use cases can be analysed at different granularity levels by separating these use cases into scenarios thereby providing a deeper understanding of the system being developed. Weidenhaupt et al. [2] showed that scenarios depict specific usage instances between actors and a system. This allows evaluation of individual, composite and interacting scenarios that specify user requirements at a granularity level that details the system's behaviour.

Real-time software systems consist of event driven processes and their requirements can be classified into behavioural requirements and temporal requirements [3]. Behavioural requirements specify the functionality of the system and temporal requirements specify timing constraints of responses from the system to specific events. Since behavioural requirements of a real-time system are descriptions of system functionality, they are easily identified. Temporal requirements, however, may not be read directly from requirements as several interacting events may define a single temporal requirement. Thus, expanding the use cases into specific scenarios, and analysing them allows analysts to clearly understand the nature of the system to be built.

This paper evaluates the suitability of scenario-based requirements analysis for real-time systems. The study

applies the methods to an example problem and discusses the implications of the results. Section 2 discusses the different scenario-based analysis methods used in this study. Section 3 discusses the case study example. It defines the research problem and then applies the chosen scenario-based models. The strengths and weaknesses of these methods are discussed and improvements are suggested where applicable. Section 4lists the contributions of this work and provides suggestions for further research on this topic.

2 Scenario-based requirements analysis methods

Several scenario-based methods have been discussed in the recent literature. This section summarises the essential elements of the scenario-based methods that have been chosen in our study.

2.1 Use case (natural language) approach

The most popular scenario-based approach that belongs to this category is use cases. As part of the UML standard, use cases are used for eliciting and analysing functional requirements [1, 4]. Use cases can be decomposed into scenarios and described using natural language. This approach is meaningful during the initial requirements elicitation phase. It simplifies the elicitation and validation due to the use of natural language and for interaction with the end users. However, this method does not provide the detail aspects of a real-time system such as timeliness, resource usage and state oriented behaviour to analysts and designers. In addition, Glintz [5,6,7] argues that use cases alone are insufficient to provide functional requirements without modelling any persistent state. Therefore, natural-language-based processes are good for defining business processes and fall short for analysing specifications for real-time systems and thus we focus on the representations briefly discussed in the following sub-section. Table 1 (in Sect. 3), however, includes a column about the strengths and weaknesses of natural language descriptions.

2.2 State chart representation

Glintz [5,6,7] presents an approach to representing scenarios using state charts [8]. This approach provides the means to represent single as well as composite and abstract scenarios. Single scenarios can be illustrated as structured text and state charts. Structured text also provides for iteration, repetition and distinction between user inputs and system responses. This allows for hierarchical as well as detailed level views of the scenarios resulting in a structure that can be analysed for real-time system attributes. Figure 1 shows the general model [5,6,7] of state chart representation. Scenarios are validated using common state chart validation methods.

Table 1 A comparative evaluation of different methodologies

	Natural language	State chart representation	Hierarchical graphical notation	Use case maps	Timed automata
Basic concepts	Uses cases, actors, pre/post-conditions, ''extends'', scenario	State charts to represent events and transitions; scenarios	Use case, actor, sequence, exception, interrupts, repetition, alternatives, scenarios	Use case, maps, pre/post-conditions, triggers, scenarios	Time-automata, constraints, operation semantics
Notation	Natural language, arrows and ovals depict relations	State charts represent hierarchical and disjoint scenarios	Graphical notation depict scenarios at environment, structural, event levels	Visual notations link responsibilities of use cases, progression of scenarios	Timed automata as spec language shows scenarios as sequence of operations
RE role	Describing requirements, finding objects	Structuring scenarios	Representing scenarios at event level	Capture behavioural relationships	Represent operational view of scenarios; provide more accuracy
Complexity	Relatively simple to read and understand	Relatively simple to write and interpret	Hierarchical decomposition provides better management	Relatively simple to read and understand	Not so simple to write and interpret
Granularity	Little support; can be ambiguous	Scenarios can be represented at various levels of abstraction; lives with ambiguity	Hierarchical models allow for successive refinement	Resource and relative temporal properties of scenarios not shown	Very detailed in describing temporal relationships between events
Focus on system behaviour	System behaviour defined in textual format	Shows abstractions, normal, and alternative scenario paths	Shows dependency between scenarios and proprieties of operating environmental	Shows relationship between functional components, behaviours, organisation	System behaviour described in scenarios as a set of time traces
Real-time support	No support, imprecise	Does not define temporal properties; can be extended for real-time systems	Supports real-time systems	Provides visibility into the behaviour of real-time system; some limitations	Very effective

Regnell et al. [9] extended the use case model with graphical notations to represent scenarios at different levels of abstractions. These levels are: environment level, structural level and event level. At the environment level, use cases are represented by relating the actors with their goals for satisfying functional requirements. The structural level describes a complete use case as a sequence of episodes that contain sequences, exceptions, interruptions and repetitions and the event level orders events in each episode as message sequence charts [10]. Figure 2 shows the three levels of representation [9].

In Fig. 2 use case X at the environment level is chosen for elaboration as a scenario at the structural level. This scenario is shown as a graph of connected episodes with time in the graph progressing downward. Regnell et al. [9] defined notations to show exceptions, repetitions and interruptions as episodes with the type identifier of the episode marked in the top left corner of the episode box. Figure 3, from Regnell et al. [9], shows an episode structure with exceptions, repetitions and interruptions. Interruptions are not connected with the flow of episodes as they can occur anywhere. They are shown next to the main episode diagram. Initiation and termination of a scenario is shown by hexagons labelled with the name of the pre and post conditions at the top and bottom of the episode respectively. Exception and interruption episodes also terminate scenarios. Event level diagrams show each episode in a scenario as a message sequence chart. Events are shown on the event level diagram encapsulated by their episodes, which illustrate the dependencies between each scenario and its operating environment properties.

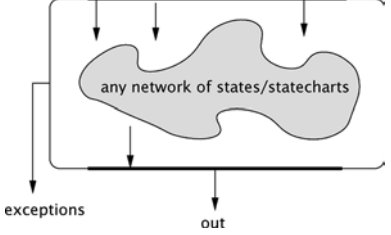


Fig. 1 State chart representation of a scenario

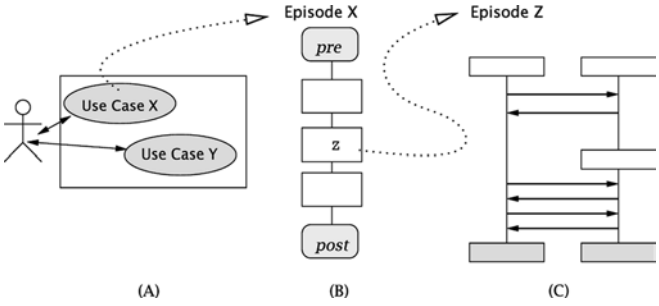


Fig. 2 Hierarchical use case models with graphical representation: Environment level (A), Structural level (B), Event level (C)

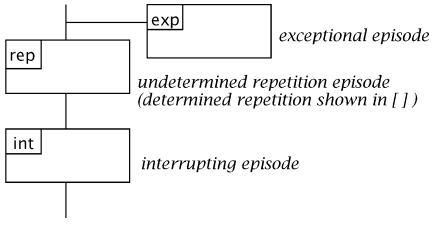


Fig. 3 Event and structural level representation of episodes

2.4 Use case maps

Use case maps [11] capture behavioural and temporal requirements of software systems. They provide visual notations to causally link responsibilities of one or more use cases. These relationships are then superimposed on abstract components that represent the structure of the system. Components can either represent software or hardware elements. This notation shows the progression of a scenario along use cases. This allows requirements analysts to reason about relationships between functional components, their behaviours, and structural organisation of the system in an explicit manner. Thus, the paths through the maps allow analysis of the system [11].

Figure 4 shows an example of a use case map for a specific scenario, along with some of the primitives available for describing a scenario path through the system. A scenario path traverses the system through components A and B. The scenario path executes up to component A, and waits for an event to join its path before proceeding to completion through component B. Filled circles represent starting points or waiting places for a stimuli to begin new scenarios and bars depict the beginning and end of paths, or marks where concurrent path segments end or begin. The filled circles in Fig. 4 represent a wait on other events or scenarios. These representations allow use case maps to show scenario dependencies and interactions and allows reasoning about scenarios.

2.5 Timed automata approach

It is important that the requirements language is simple enough to be easily understood, but expressive enough to fully describe the desired requirements. Formal methods of requirements specification came into being as a result of the lack of precision and presence of ambiguity in narrative requirements specifications. They have been used to improve the quality of software specifications by modelling and formalising the requirements of a system [12, 13].

Formal methods are mathematically based techniques, which work as a fault-avoidance techniques that can increase dependability by removing errors at the

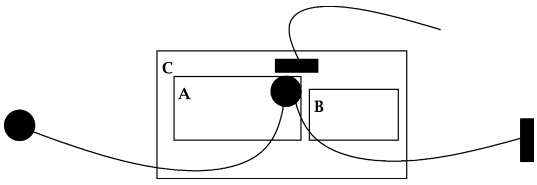


Fig. 4 Use case map description showing scenario coupling

requirements, specification and design stages of development. Another main advantage in using this approach is the ability to structure a set of scenarios.

The main motivation for using formal methods is to ensure the quality of the system by adhering to the following attributes:

- *Ambiguity:* formal methods avoid ambiguity by making a requirement subject to only one interpretation.
- *Completeness:*formal methods achieve completeness by describing all significant requirements of concern to the user, including requirements associated with functionality, performance, design constraints, attributes, or external interfaces.
- *Consistency:*formal methods make requirements to be consistent by avoiding conflicts among requirements.
- *Verifiability:*formal methods provide a finite and cost-effective process with which a developed software system can be verified if it meets the requirements.
- *Modifiability:*formal methods ensure that any changes to the requirements can be made easily, completely, and consistently, while retaining the existing structure and style of the set.

On the other hand, formal methods are not widely visible for the following limitations:

- Formal specifications are difficult to read and understand
- Formal methods cannot help model all aspects of the real world
- Correctness proofs are resource-intensive
- Development costs increase (for some companies and projects)
- Formal specifications can still have errors

Timed automata Formal methods can be used effectively in the specifications of real-time systems but this advantage comes at the expense of the readability and effort to write the scenario. Therefore, for simplicity, this paper will consider a semi-formal representation called *timed automata*.

All scenarios described earlier were proven useful to some degree in requirements engineering. The timed automata approach, as described in Some et al. [14], applies timed automata to scenarios with timing constrains, which provides an accurate way of considering user requirements. It uses timed automata as a target specification language. This approach represents the operational view of scenarios with the support of a semi-formal representation that is needed for real-time systems. It uses operations semantics, and a mapping between concepts of scenarios, and those of the theory of timed automata. The major advantage of this approach is its accuracy, simplicity and readability.

A scenario using the timed automata approach is composed of interactions and reactions triggered by a series of operations and conditions (stimuli). The time of occurrence of scenarios can be constrained by delays and timeouts. An interaction delay specifies the maximum, minimum or exact amount of time that must be maintained between operations. The timeouts specify the maximum delay for completion of an interaction or scenario. Scenarios that involve timing constrains can be formally represented using the following global constrains: $R_{\text{initial–delay}}$, $R_{\text{pre–cond}}$, R_{timeout}, $R_{\text{repetitions}}$. A scenario is composed of all possible sets of timed traces $(op_1, \delta_1)...(op_n, \delta_n)$ where each op_i is an operation and i is the instant it occurs according to an abstract clock. δ_i is the clock variable that corresponds to the operation op_i, which will be initialised and associated with different transition to the state of the next operation [14].

The time of occurrence of operations can be constrained by initial interaction delays, the operation's timeout, and the scenario's timeout. Expiry operations may be associated with timeouts in order to be executed when the maximum delay passes. Figure 5 shows an example of scenario represented using the timed automata approach. This example shows the scenario as a sequence of operations. Each operation maintains a clock variable that corresponds to the operation. Each operation might have temporal constraints or conditions and it occurs only if they hold.

2.6 Rationale for model selection

We will focus on the four models that were briefly described in previous sub-sections.

Real-time systems are event driven, periodic and are differentiated from other software systems due to its

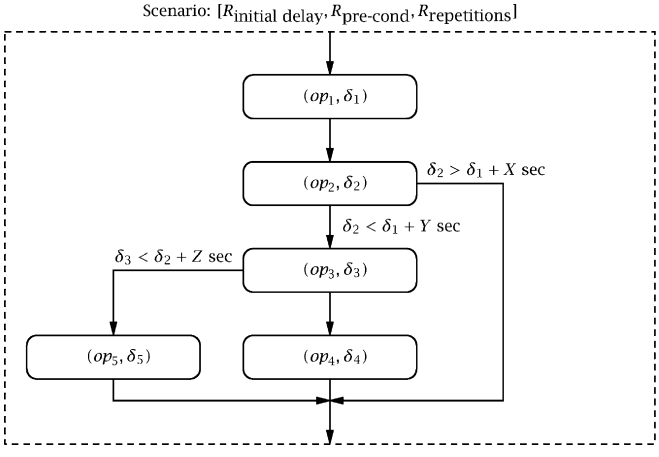


Fig. 5 Timed automata representation of a scenario

timeliness and throughput characteristics. Also in these systems, more than one event could be active at any given time providing the notion of concurrency. In addition, any requirements analysis technique needs to provide for an unambiguous and consistent reading of the requirements. Another attribute is that these requirements need to be traceable to hierarchical user requirements. Therefore the models we have chosen to compare had to incorporate the concept of events, concurrency, timeliness and abstraction. In addition, techniques that supported visual notations were given preference since they were easier to read, analyse and automate than natural language or mathematical model based techniques. The four models we choose to study provided powerful visual notations to illustrate and reason about scenarios. These models also supported real-time characteristics such as event and time representation and concurrency. Finally, as discussed above, the concept of abstraction is provided for in all these models thus facilitating the study of these requirements at different levels of hierarchy.

To summarise, these models were selected because:

- The selected models provide visual notations and are easier to interpret.
- The selected models support representation of scenario interaction facilitating the study of concurrency in real-time systems.
- The selected models also support real-time characteristics such as event and time representation and concurrency.
- The selected models provide support for abstraction, thus facilitating the study of the requirements at different levels of hierarchy.

Evaluation Criteria The chosen models will be evaluated based on the following criteria:

- *A model's representational ability of a real-time systems characteristics, such as temporal requirements, concurrency and asynchronous and periodic events.*
  A real-time software system's essential characteristic is that all actions performed by the software meet the time specifications stated by the requirements. Also, real-time software systems contain periodic and asynchronous events that influence temporal and functional specifications. In order to analyse the requirements, the scenario-based method should provide a mechanism to specify time and events in scenarios.
- *Analysis at different levels of abstraction.*
  It is well known that complex software systems are easier to analyse and understand when they are decomposed in a hierarchal manner. It can be said that real-time software systems are more complex since they have to fulfil functional as well as many non-functional requirements, such as response time and throughput. Hierarchal decomposition allows software architects and software designers to focus on the analysis at their levels of granularity. For example architects will use the analysis to select suitable architectures, while design engineers will select design patterns that satisfy the requirements. Therefore the ability to provide a mechanism for specifying abstraction in scenario-based analysis methods is important.
- *Representational and analytical capabilities to deal with scenario interactions.*

Requirements can seldom be represented by individual scenarios. Real-time software systems consist of many threads or processes that run concurrently and these threads and processes synchronise with each other to accomplish the requested tasks. Then in this instance many scenarios interact with each other to satisfy a requirement. Therefore a real-time scenario-based analysis technique should provide support for representing multiple scenarios and interactions among them.

The first criteria was selected as it contained the essential characteristics of a real-time system as described in next section. The second criteria was selected as it represents important requirements engineering characteristics. Since an analysis of real-time systems requirements would contain many scenarios and since real-time systems exhibit the nature of concurrency it was essential to study the capability of the techniques to deal with dependencies and interactions between scenarios.

3 Case study: remote temperature sensor (RTS)

In this paper, we discuss four of the scenario-based analysis methods previously illustrated on a real-time industrial application. This example and the associated requirements are chosen to highlight the differences between the four analysis techniques when they are studied against the following real-time attributes:

- Temporal requirements or timeliness of system responses
- Illustration of concurrent requirements
- Composition of functional and temporal needs
- Resource usage limitations that affect response and/or concurrency

3.1 Problem definition

The example chosen is a real-time industrial application illustrated by Barbacci et al. [15]. This application consists of a remote temperature sensor (RTS), 16 temperature furnaces and 16 computer hosts. The RTS consists of 16 furnaces and a digital thermometer as shown in

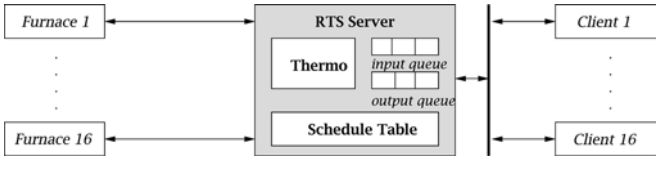


Fig. 6 Structural view of RTS application

Fig. 6 and periodically updates the hosts with furnace temperature readings. The host computers may also request individual readings in an asynchronous manner. During these requests the furnace is provided its update rate for future periodic updates. The following list of requirements are selected from Barbacci et al. [15]. The authors deem these requirements to contain aspects of timeliness, concurrency, resource usage and major system functionality:

– The hosts shall receive periodic readings from the RTS system at the times specified by each host computer.
– The hosts must receive an initial report of the furnace temperature within 10 s of sending a request.
– Furnace reading intervals shall be limited from 10 to 90 s.
– The system shall support 16 furnaces and hosts.

Our evaluation begins with the decomposition of requirements into scenarios, then illustrating them using the four different analysis techniques, followed by a discussion of each method's effectiveness for specifying real-time system requirements. The following scenarios are distilled from the requirements stated above and contain individual, composite and interacting scenarios.

1. A specific host sends a control message and waits for the initial response. RTS system responds with the temperature reading within 10 seconds, and updates the schedule table for future periodic updates.
2. The RTS system periodically updates each host at their scheduled intervals.
3. Digital thermometer is read by the RTS for a specific furnace's temperature.
4. Each host message is acknowledged by the system.

3.2 Analysis

Four models are considered for the study: (1) Closed state chart and structure text model [5], (2) Hierarchical use case model [9], (3) Use case maps [11] and (4) Timed automata approach [14].

These models were chosen for the following reasons:

– Models provided a mechanism to visualise real-time requirements such as response and throughput times and capture event driven actions
– Models contained support for representing interacting and composite requirements and thus provide an opportunity to specify concurrency between requirements
– Dependencies that affect timeliness such as resource usage can be either illustrated or superimposed on models

The scenarios defined for the example study can be composed into the two abstract scenarios listed below. This composition will hold true throughout the entire study.

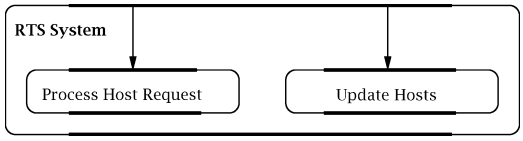


Fig. 7 High level scenarios of the RTS system

– Host request of a temperature reading from a specific furnace
– RTS system's periodic update to all hosts

3.2.1 State chart representation

Using Glintz's state chart model [5] the two abstract scenarios are shown in Fig. 7. The Process Host Request scenario begins when an incoming message is detected by the RTS system via an interrupt event. The message is analysed by the system and a message acceptance or rejection acknowledgment is sent to the host. If the message is accepted then the schedule table is updated and the corresponding temperature furnace is read and forwarded to the host.

Glintz's method [5] allows for abstraction of elementary scenarios into a composite abstract scenario. The Process Host Request abstract scenario is composed of three elementary scenarios: Analyze Message, Read Furnace Temperature and Send Message. The state chart of Fig. 8 shows the transitions between these abstract scenarios and the interconnections and dependencies of scenarios based on the events in the system. These events are also denoted with conditions under which they move to the next causal scenario. In Fig. 8, the system moves to the Read Furnace Temperature scenario when a valid message is detected. If the message is not valid, the system moves to the Send Message scenario. The Read Furnace Temperature scenario reads the temperature from the specified furnace and provides the temperature to the system. Glintz's method [5] also allows description of each scenario in a structured text format as shown below:

Type Scenario: Analyze Message

Actor: RTS System

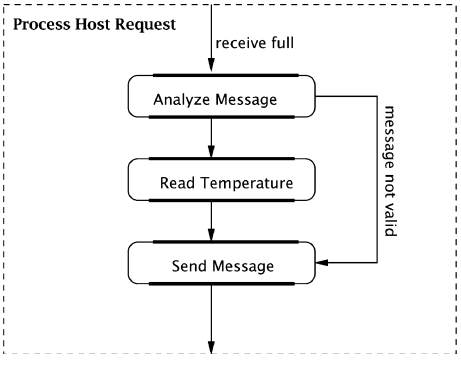


Fig. 8 State chart of process host request scenario in RTS

Normal Flow:

1. System: Host message received.
2. System: Start request timeout timer.
3. System: Read message.

4.1. System: Validate message.

4.2. Identify furnace number and update furnace period.

5. Request temperature reading.
6. Send temperature reading to host.

Alternative Flow:

4.1'System System validates message.

– **if** message is invalid then return error message, **terminate; endif**.

3',4',5' and 6' System System sends temperature reading

– **if** request timer times-out then send error message, **terminate; endif**.

In the above instance the state charts were able to specify the detail functionality of the application using elementary and composite scenarios. They were also able to show the dependencies between scenarios. However, the temporal specifications of 10 s for request completion was missing in the model. One can argue that the structure text description shown above captures this requirement, illustrated under the alternative flow section. Since the timeout can occur in any one of the normal flow steps it would have to be listed as an alternative for each normal flow step. This reduces readability and affects the temporal analysis of the system.

Figure 9 illustrates the Update Hosts scenario. Updates are performed periodically at the intervals specified for each host. This scenario begins when the scheduled update time expires for each host and is then followed by a temperature reading from a specified furnace described by the Read Furnace Temperature scenario. The same scenario is used to read the temperature from all sixteen furnaces. Therefore the composite scenario Update Hosts repeats each time the timer expires for every furnace listed in the schedule table. This information is difficult to represent directly in the state chart model, but could be listed in the structured text that follows it.

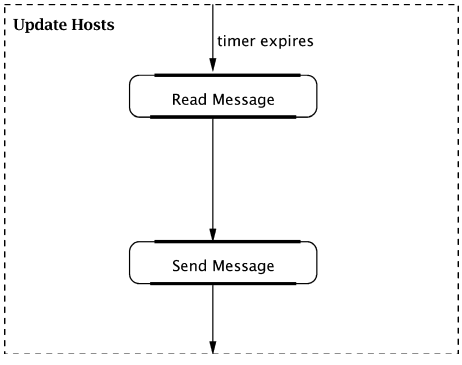


Fig. 9 State chart of update hosts scenario in RTS

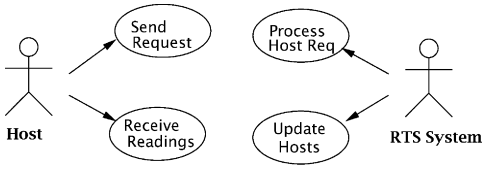


Fig. 10 Summary level use cases of RTS system

3.2.2 Hierarchical graphical notation

The second method we study for describing scenario-based requirements is the traditional use case model extended with additional graphical notations and hierarchical views proposed by Regnell et al. [9]. This notation supports representing requirements at the environment, structural and event levels. The environment level is described by conventional use case diagrams [1] as shown in Fig. 10. These use cases represent the high level scenarios in our evaluation. These high level scenarios, Process Host Request and Update Hosts are further refined into structural and event levels.

The structural level description of use case Process Host Request is shown in Fig. 11. There are four episodes in this use case: (1) analyze message, (2) send error message, (3) read furnace, and (4) received message. A message is received by the RTS and is informed of this message through an interruption. If the message is valid then the scenario progresses to the next causal episode read furnace, else the scenario branches to the exception episode send error message and terminates. We consider

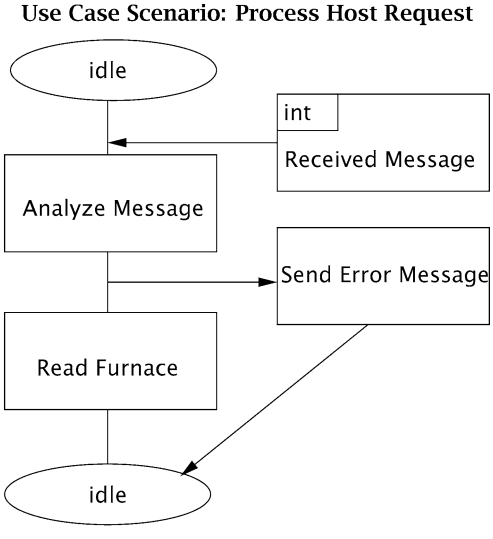


Fig. 11 Structural level description of scenario process host request

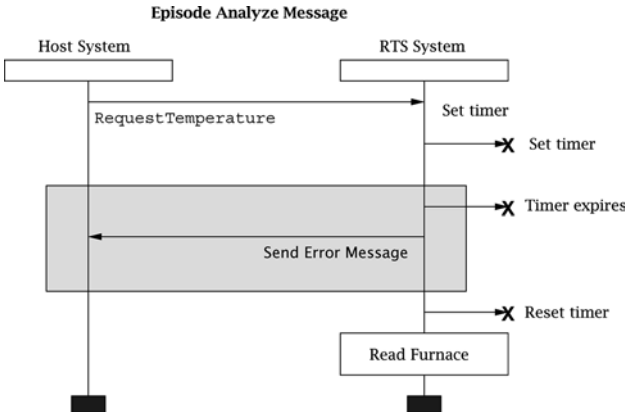


Fig. 12 Event level description of scenario process host request

episode analyse message for further decomposition at the event level in Fig. 12. In this figure, a host requests a temperature reading from its respective furnace. The RTS system responds to this request. The temporal requirements of the request are then shown by a timer activation point, a timeout point and a timer reset point in the event level diagram. This shows the reader that this is a time dependent scenario and failure to meet the timeliness requirement would end in the generation of an error message. In Fig. 12 the event level diagram coveys the following information about the system:

- Episode Analyze Message is bounded by response time constraints.
- Episode Analyze Message can either generate an exception or proceed to its next causal episode Read Furnace.

Figures 13 and 14 represent the structural and event level diagrams of the Update Hosts scenario. Episodes included in this scenario are: read furnace and send temperature. The scenario begins with an interrupting episode that requests a temperature reading from a specified furnace. The interrupting episode is triggered when system time is a multiple of any one of the times in the schedule table. Once the temperature is read, the RTS system processes the information and forwards the reading to the appropriate host. This completes the update for the specific host. However the requirement states that the system update all sixteen hosts in a periodic manner. This information is not depicted at the structural level. One can provide the argument that placing a repetition operator in the read temperature episode could satisfy this requirement. However as time progresses downward at the structural level placing this operator in the read temperature episode would mean that we are awaiting 16 consecutive interruptions (i.e., 16 read temperature requests) from the system without sending any readings back to the system. Clearly this would be an incorrect specification at the structural level.

The event level diagram of Fig. 14 shows the temporal requirements of episode Read Temperature. The event begins with a temperature request to the furnace. The thermometer entity reads the temperature from the requested furnace and forwards the data to the RTS. Since our requirements state that the consecutive furnace readings be at least 10 s apart and no more than 90 s apart the message has a conditional parameter: (last reading time + 10 s) < current time < (last reading time + 90 s). This indicates a wait on the furnace reading and represents a resource limitation.

Therefore, we have shown that the extended graphical and hierarchical views proposed by Regnell et al. [9] allows representation of temporal and resource usage. However the representation fell short in illustrating how single episodes could be used multiple times at the event level or structural level.

3.2.3 Use case maps

The next method chosen for evaluation is use case maps. We have chosen the essential primitives of use case maps to illustrate our example. Use case maps show related scenarios or use cases in a map-like diagram. Each scenario in a use case can be shown as a path through the problem space that traverses structural components [11] of the system. These paths traverse black box compo-

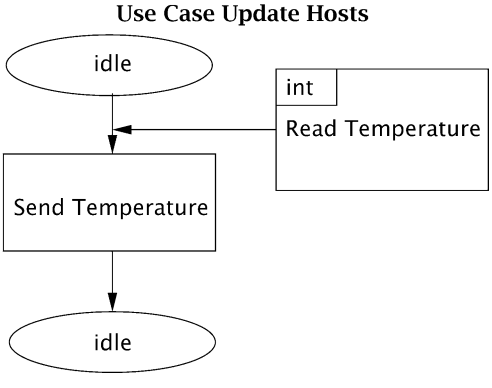


Fig. 13 Structural level description of scenario update hosts

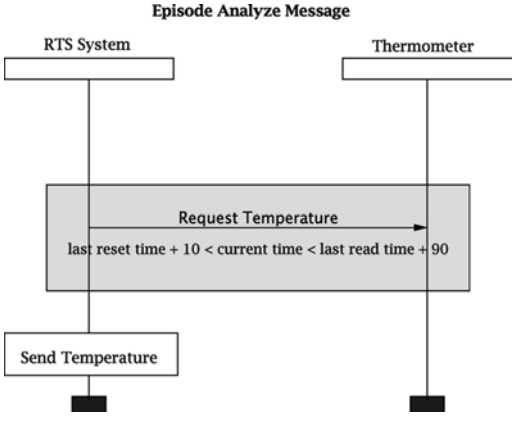


Fig. 14 Event level description of episode read temperature

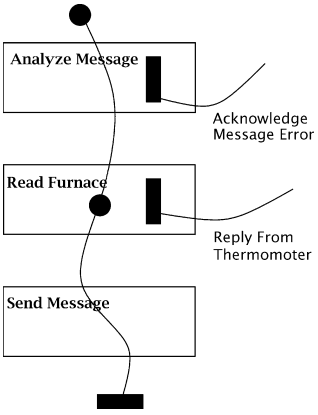


Fig. 15 Use case map for process host request scenario

nents of a system. Figure 15 represents the use case map for Process Host Request. The use case path encounters a fork at the Analyze Message component that dictates the occurrence of an alternative flow if the request message is not successful and a wait condition in the path at the Read Furnace component shows that the scenario cannot move forward until a reply from the thermometer is received. Structural components, forks and wait states then allow representation of behaviour, interactions and resource limitations. Then this representation could be used for analysing high-level design decisions and organising the components for handling interactions and concurrency in the system. However, elementary notations in use case maps are unable to represent repetitions of use cases from a system level perspective, and are limited in that sense.

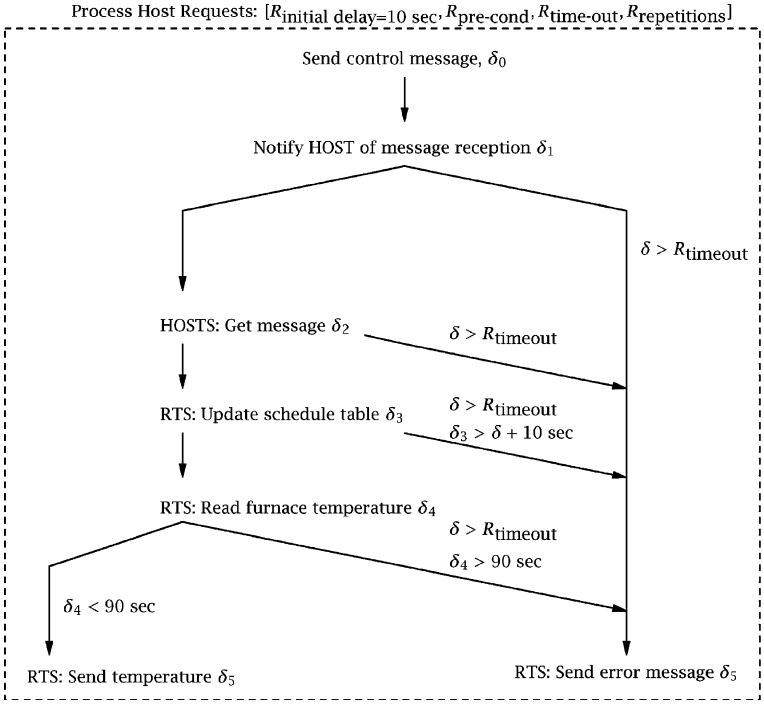
3.2.4 Timed automata

This new formalised view of representing scenarios using timed automata, scenarios can be constrained by delays and timeouts. Figures 16 and 17 show how timed automata models the two abstract scenarios that have been identified in the RTS system. Each scenario was represented as a sequence of operations.

In Fig. 16, the Process Host Request scenario begins with checking for a pre-condition or set of pre-conditions that must hold in the system prior to the scenario execution, some temporal constrains (like number of trials and timeout). Then, the Process Host Request scenario executes next operations in the following order: (1) Any Host can send control messages to RTS system requesting for a Furnace temperature, (2) RTS system notifies Host with the reception of the message, (3) Host receives the confirmation of the reception, (4) RTS system updates the schedule table with schedule intervals for that Host, (5) RTS reads the Furnace temperature, and then either (6) RTS sends temperature to Host if waiting time is less than 90 s or (7) RTS sends error message to Host if waiting time is more than 90 s. Each operation maintains a clock variable δ_i that corresponds to the operation and monitors the abstract clock δ for scenario timeouts. Any operation is dependent on meeting time constrains of the previous operation δ_{i-1} and the abstract clock, i.e., The abstract clock $= \sum_{i=1}^{n} \delta_i$.

Similarly, Fig. 17 describes the Update Hosts scenario. Updates are performed periodically at the intervals specified for each host. This scenario begins after scheduled update time (initial delay) expires for the host. Then, it is followed by a temperature request from

Fig. 16 Process host request scenario using timed automata



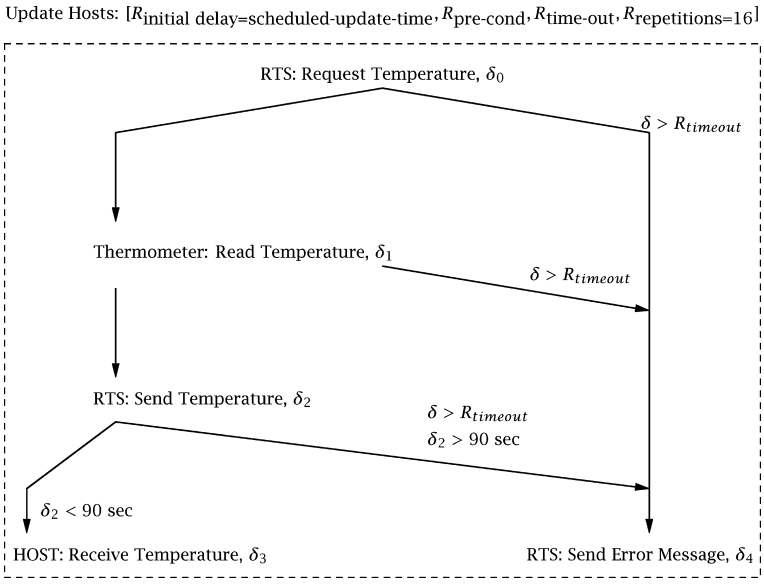


Fig. 17 Update hosts scenario using timed automata

a specified furnace described by the Read Furnace Temperature scenario. This scenario is also called by Process Host Request to read the temperature from the furnaces.

This approach can be combined with any previous scenario approach to represent requirements specifications at different levels of abstractions in a more accurate way. Figures 18, 19 and 20 show some examples of how the timed automata approach can be applied to different scenario approaches described earlier to make them more formalised and accurate. Clearly, timed automata approach is accurate, very simple to read and understand, and provides the support of formal representation that is needed for real-time systems.

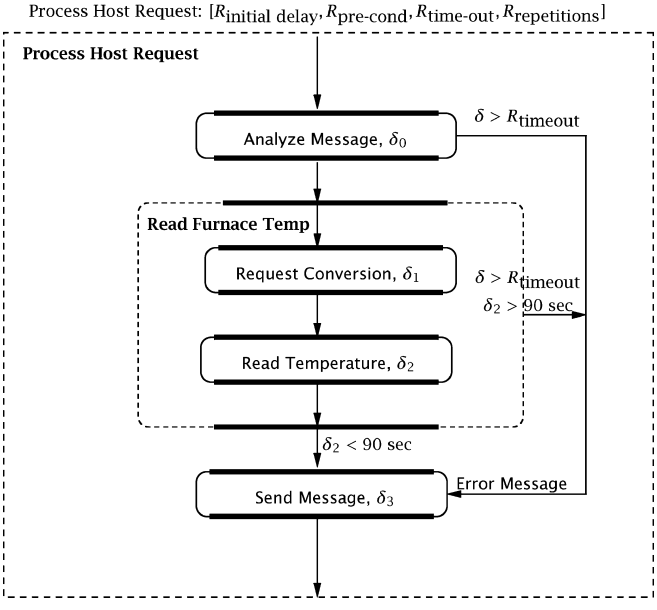


Fig. 18 Applying timed automata to state chart of process host request scenario

3.3 Discussion

We have applied four different scenario-based models to a real-time software system. We analysed two essential requirements of the system, which were composed of many elementary scenarios.

The state chart and structure text model was able to capture the behavioural nature of the system. Temporal and resource usage elements were difficult to specify visually, but were structured into a formal text. Composition of scenarios allowed the reader to show concurrency between requirements.

The extended hierarchical use case model illustrated the scenarios at three levels: environment, structural and event level. The behaviour of the system was illustrated at the structural level and the temporal and resource usage information was shown at the event level. Concurrency was not explicitly shown except as use cases.

One δ of this model is the need to provide tool support, not only for diagram drawing, but also for automated analysis and checking. It would be useful to use formal language notation for system action specifica-

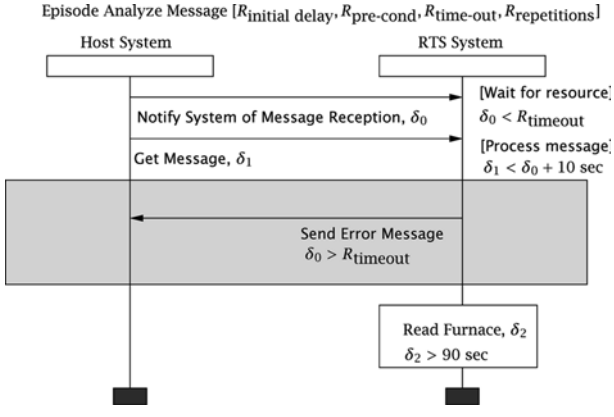


Fig. 19 Applying timed automata to hierarchical graphical notation for event level of process host request scenario

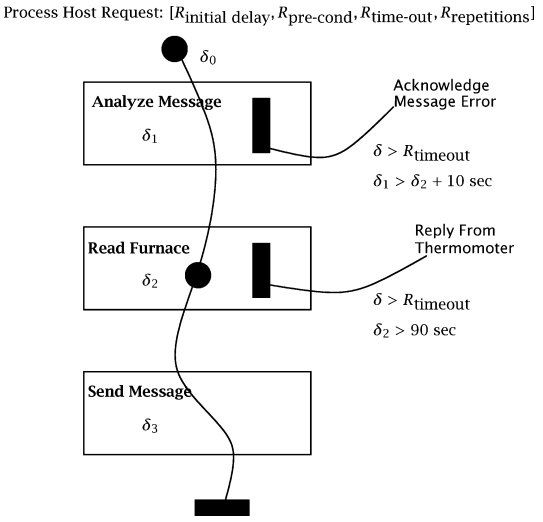


Fig. 20 Applying timed automata to use case map for process host request scenario

tions. In other words, there is a need to formally define the syntax and semantics of the presented use case model in some metadata language, which is an improvement that was handled using the Timed Automata approach.

Buhr and Casselman [11] use case maps provide a road map of scenario execution. They show the interacting and dependent nature of all elementary scenarios involved in the completion of the abstract scenario Process Host Request. Resource and relative temporal properties of the scenarios are not shown. Most event driven events such as interrupts can be shown as forks in the path, however, assigning temporal properties to these paths are not possible. This prevents a complete analysis of the system.

The timed automata approach aimed to formalise scenarios in a simple, readable, and accurate way to consider users requirements. Also, it describes the temporal relations between events. The formalism developed can be used to build an algorithm that generates timed specifications. This approach can be improved by using more operation semantics, and mapping between concepts of scenarios, and those of the theory of timed automata.

Scenario-based analysis of real-time software requirements provide a level of granularity desired by analysts and developers to specify and define a real-time software system. Findings from this work are:

- State charts along with structured text provides behaviour level analysis of the system and can be extended to represent temporal properties. Concurrent scenarios can be shown as abstractions at a higher level. The model does not show any support for resource usage.
- Message chart or sequence based scenario illustrations provide a more complete mechanism to analyse real-time systems. The annotation of time, sequences of scenario interactions and resource usage limitations are facilitated by this method.
- Use case maps provide a good overall view of all scenario dependencies and interactions between components present in the system. Concurrency can be represented but temporal properties are limited only if elementary use case map constructs are used. We consider this method as a good first starting point for identifying the relationships between scenarios, followed by message sequence charts of each individual scenario for a complete specification.
- The timed automata approach can be combined with any of the discussed scenario approaches to represent requirements specifications at different levels of abstractions in a formal and accurate way. The timed automata approach can improve the weaknesses of the extended hierarchical use case model in defining the syntax and semantics of the model more formally. This will provide the support of a formal representation that is needed for real-time systems.

In order to evaluate and analyse real-time systems, a comprehensive understanding of various techniques and approaches is necessary. Table 1summarises major scenario approaches covered in this paper for real software systems using different representations. The table comparatively illustrates different scenario approaches based on various categories that investigate the validity of an approach to support real-time systems.

The categories used in the comparison address the most important attributes that corresponds to the chosen evaluation criteria. Some categories used in the evaluation such as ''Focus on System Behaviour'' and ''Real-Time Support'' will check the model's ability to represent real-time systems characteristics. Other categories such as ''Complexity,'' ''Granularity'' and ''Requirements Engineering Role'' will measure the model's ability to analyse requirements at different levels of abstraction and discuss its representational capabilities to deal with scenario interactions.

Table 1 also shows that some approaches satisfy certain evaluation criteria for real-time systems but not all required attributes. For example, the state chart representation can represent scenarios at various levels of abstractions but it doesn't define temporal properties of the system. Also, hierarchical graphical notation shows dependency between scenarios and represents scenarios at different hierarchical levels with some limitations in showing some temporal properties of scenarios. This comparative illustration helps in identifying the strengths and weaknesses of each approach which leads to finding ways to improve each approach for different software systems.

An approach such as timed automata can be combined with other scenarios such as state chart representation or use case maps to construct a model that is able to represent real-systems characteristics and represent requirements specifications at different levels of abstraction.

Our key argument is that an approach that satisfies our evaluation criteria will be the most effective in analysing and evaluating real-time systems.

4 Conclusions

In this paper, we have analysed and evaluated some methods for analysing scenario-based requirements of real-time systems. Real-time systems are characterised by temporal and resource constraints in addition to user needs. Our study indicates that the message sequence charts used by Regnell et al. [9] comes close to depicting a real-time system for complete scenario-based analysis when it is combined with the timed automata approach. This new approach will define the syntax and semantics of the model more formally and provides the support of a formal representation that is needed for real-time systems. The use of three different levels in Regnell et al. [9] model allows the analyst to view and reason about the systems responses to the scenarios from different perspectives thus considering the needs of all stakeholders in a system.

Real-time software systems are characterised by response times, periodic and asynchronous events and resource constraints. Therefore any real-time requirements analysis technique should attempt to include these characteristics in their methods. We have discussed four such methods above and now provide our recommendations.

State chart and message sequence chart illustrations allow abstraction and thus allows stake holders of the system to analyse the system at their level of granularity, which enhances understanding of the system. Although state charts are able to represent scenarios and their dependencies to various events there were no provisions for showing time dependencies of the scenario elements. However, when the state chart model is combined with the timed automata approach, as shown in Fig. 18, time dependencies can be then represented allowing a more thorough and accurate analysis. Message sequence charts allow the representation of time, events and scenario interactions. However the semantics for representing time are very elementary and does not provide support for analysing the temporal properties at the macro level. Extending the message sequence charts with the timed automata approach provided a rich set of semantics and relationships to an abstract clock that could be used for real-time analysis. Finally, message sequence charts allowed the manipulation of hardware or system resources as shown in Fig. 12 and the state chart method did not provide semantics for such notations. Therefore we conclude that extending message sequence charts with the timed automata approach satisfies representation and analysis of all real-time characteristics.

Our work was applied to a uni-processor environment and thus was devoid of any true multi-threaded scenarios or concurrent scenarios. Another limitation of our study is the extent to which use case maps were used in scenario analysis. It is well known that use case maps not only provide the means for scenario or requirements analysis but also provide the bridge between requirements analysis and design [11, 16]. Thus use case maps contain a rich set of notations to annotate interrupt service requests, processes and threads. We have limited our use of the notations to paths, forks and joins and wait states. Finally this work can be extended to multi-processor based environments to investigate the models effectiveness in specifying concurrent scenario-based systems. Another interesting application would be to use the complete use case maps notation for scenario-based analysis of real-time systems.

References

1. Jacobson I, Christerson M, Jonsson P, Overgaard, G (1992) Object-oriented software engineering–a use case driven approach. Addison-Wesley, Reading, MA
2. Weidenhaupt K, Pohl K, Jarke M, Haumer P (1998) Scenarios in system development: current practice. IEEESoftw 15(2):34–45
3. Ekelin C, Jonsson J (1996) Real-time system constraints: Where do they come from and where do they go? In: Proceedings of the International Workshop on Real-Time Constraints, Alexandria, VA, October 1999, pp 53-57
4. Amyot D, Mussbacher G (2000) On the extension of UML with use case maps concepts. In: Proceedings of UML2000, York, UK, October 2000. Lecture notes in computer science, vol 1939. Springer, Berlin Heidelberg New York, pp 16-31
5. Glinz M (2000) Improving the quality of requirements with scenarios. In: Proceedings of the Second World Congress for Software Quality (2WCSQ), Yokohama, September 2000, pp 55–60
6. Glinz M (2000 Problems and deficiencies of UML as a requirements specification language. Proceedings of the 10th International Workshop on Software Specifications and Design (IWSSD-10), San Diego, November 2000, pp 11–22
7. Glinz M (2000) A lightweight approach to consistency of scenarios and class models. In: Proceedings 4th IEEE International Conference on Requirements Engineering. Schaumburg, IL, June 2000, pp 49–58
8. Harel D (1987) StateCharts: a visual formalism for complex systems. SciComput Program 8:231–274
9. Regnell B, Andersson M, Bergstrand J (1996) A hierarchical use case model with graphical representation. In: Proceedings of ECBS'96, IEEE International Symposium and Workshop on Engineering of Computer-Based Systems, March 1996
10. Message Sequence Chart (MSC) (1993) ITU-T Recommendation Z.120, International Telecommunication Union
11. Buhr R, Casselman R (1996) Use case maps for object-oriented systems. Prentice Hall
12. Hsia P, Samuel J, Gao J, Kung D (1994) Formal approach to scenario analysis. IEEESoftw 11(2):33-41
13. Leveson N, Heimdhal H, Hildreth H, Reese J (1994) Requirements specifications for process-control systems: lessons learned and steps to the future. IEEETransSoftwEng 20(9):684-706
14. Some S, Dssouli R, Vaucher J (1995) From scenarios to timed automata: building specifications from users requirements. Asia Pacific Software Engineering Conference, December 1995
15. Barbacci M, Carriere S, Feller P, Klien M, Kazman R, Lipson H, Longstaff T, Weinstock C (1998) Steps in an architecture tradeoff analysis method: quality attribute models and analysis. Technical Report TR-97–029. Software Engineering Institute, Pittsburgh, PA
16. Saiedian H, Dale R (2000) Requirements engineering: making the connection between the software developer and customer. Information and Software Technology 42(4):419–428