High-Level Representation of Time in Diagrammatic Specification

Sabah Al-Fedaghi*

Abstract

The notion of time is an important element in such systems as real-time embedded systems. Real-time systems have strict timing constraints, and their complexity is continuously increasing, making their design very challenging. This paper concerns a very high level of requirements specification used for system understanding and communication among stakeholders and as a base for development. It introduces a diagrammatic description of functional behavior of a system with nonfunctional constraints including timing plan. Specifically, this paper explores the presentation of time at this level of system description. The usability and feasibility of the proposed method are illustrated by applying it to examples.

Keywords: Time representation; model-based methodology; conceptual modeling; diagrammatic specification of systems

1. Introduction

The notion of time is an important element in such systems as real-time embedded systems, especially if critical features (e.g., safety) are functionally required. An embedded system is a system that interacts continuously with its physical sphere via sensors and actuators. Unique issues of these systems are how to represent time, how to capture causality behavior, and how to integrate functional and timing activities. Time-constrained behavior of an embedded system leads to the need to address issues of system predictability and dependability in addition to functional correctness.

* Corresponding author. Tel.: +96599939594; fax: +96524839461.
E-mail address: sabah.alfedaghi@ku.edu.kw
Real-time systems have strict timing constraints, and their “complexity is continuously increasing which makes their design very challenging”\(^3\). To simplify this complexity of systems development, in general, and specifically in the field of embedded systems, new paradigms have been proposed such as model-based methodologies as a holistic top-down approach, correlating specifications and design models\(^2\). In model-based development, overall system development is abstracted at several levels to support the design, verification, and analysis techniques of the system.

According to Gherbi and Khendek\(^3\), “UML is suitable to deal with complexity” in real-time systems. In UML, timing diagrams focus on the time of events that cause changes in lifelines. They typically include lifeline, state or condition timeline, destruction event, duration constraint, and time constraint. A timing diagram may show states of the attribute, or testable conditions, such as a discrete value of an attribute. Duration constraint refers to a duration interval used to determine whether the constraint is satisfied, e.g., passage of a certain length of time.

Nevertheless, an underlying tool for expressing the unified totality of a system’s processes and concepts is lacking in UML. Heterogeneous and multiple diagrams are typically described by informal text-based description. According to Bock\(^4\), “Requirements modelling includes the translation of textually expressed needs into a computable form, … [In translation of text to model phase] requirements usually appear first as a large text document, structured in some way by headings.” In general, specification models, e.g., UML diagrams, and analysis-level models, e.g., architecture description languages (ADLs), are supposed to be woven together on a ground fabric of this text-based description.

This paper proposes a diagrammatic model to serve as an alternative semantic ground fabric for different types of applications, e.g., requirements specification, time constraints, software development, security binding, contract conditions, shared understanding, and documentation. The model itself can be utilized at different abstraction levels.

2. Some related work

Timing is typically incorporated after tasks and software architectures are defined, when holistic scheduling algorithms and expected worst-case execution times are analyzed\(^5\). This holistic schedulability is integrated with timing analysis of hard real-time messages applied a priori to a configuration to determine the timing of the system as a whole\(^5\). The schedulability analysis for tasks is described by an equation to compute the worst-case response time of a given task.

This paper does not involve such a detailed level of description; rather it is concerned with a very high level of requirements specification, e.g., the level of UML use-case, sequence, and activity diagrams. It is developed for early system understanding and communication among stakeholders, including those without technical knowledge, and facilitates agreement with clients/users, and can be used as a base for system development. The paper introduces a diagrammatic description of the functional behavior of a system with nonfunctional constraints, including a timing plan. One of the objectives is to avoid ambiguous textual language and heterogeneous diagramming. It can also become the basis for incremental developmental design of systems. Specifically, this paper explores representation of time at this level of system specification.

3. Flowthing model

As background for constructing a representation of scheduled timing in a system, this section reviews the underlying notions used in such a task. The example in this section is a new contribution.

The Flowthing Model (FM)\(^6,7,8,9\) represents some segment of reality as a web of interrelated flows that cross boundaries of intersecting and nested spheres. For example, each user is represented by a sphere with a hierachical structure. Ingredients in a flow include flowthings (things that flow, artifacts), and their flow systems (flowsystems): a structure of flow with at most six stages (see Fig. 1). Messages, pictures, news, and opinions are examples of flowthings. A “thing” is defined as a flowthing: that which is created, released, transferred, arrived, accepted, and processed, while flowing within and among spheres. It has a permanent identity but impermanent form, e.g., the same news translated into different languages. A flowsystem constrains the trajectory of flow of flowthings. To flowthings, the flowsystem is formed from six discontinuities: being created, being released, being transferred, being arrived, being accepted, and being processed.
Flows connect six stages that are exclusive for flowthings; i.e., a flowthing can be in one and only one of these six states at a time: transfer, process, creation, release, arrival, and acceptance, analogous to water being in one of three states in Earth’s atmosphere: solid, liquid, or gas. A stage here is a “transmigration field” of the flowthing that is created, processed, and released, is transferred, arrives, and is accepted (or simply received, combining arrived and accepted into one state). In Fig. 1, irreversibility of flow is assumed, e.g., released flowthings flow only to transfer.

The exclusiveness of FM stages (i.e., a flowthing cannot be in two stages simultaneously) indicates synchronized change of the flowthing, e.g., a flowthing cannot be changed in form and sphere simultaneously. This is a basic systematic property of flowthings. Note the generality of the notion of flow in FM. For example, creation of a flowthing is a flow (from nonexistence, i.e., not currently existing in the system, to existence, i.e., appearance in the system).

Initializing, stopping, and continuing of flows occur through triggering, a control mechanism. It is the only linkage among elements in FM description besides flow and is indicated by dashed arrows. Synchronizations (e.g., join/fork) and logic notions (e.g., and/or) can be superimposed over the basic FM depiction. Note that these mechanisms can be modeled as flowthings.

**Example** (from Alspaugh10): Consider the UML sequence diagram of an ATM system shown in Fig. 2. Fig. 3 shows its corresponding FM representation.

The user first inserts his/her card, and the number flows to the ATM (circle 1 in the figure), where it is processed (circle 2) to trigger (3) the creation (4) of a request for pin number. The request flows to the user (5) to trigger (6) entering (creation) of the pin number (7) which flows to the ATM (8).
There, the pin number is processed (9) and sent (10) to the bank, where it is processed (11); if valid, it triggers the generation of a valid (12) message. The valid message flows to the ATM (13), where it is processed (14) to generate options that flow to the user (15). This causes the user to generate his/her option of withdrawal (16) that flows to the ATM machine (17), triggering (18) a release of cash that flows to the user (19).

4. Time representation in FM

4.1 Time as a flowthing

The new premise in this paper is that time is a flowthing that can be created, released, transferred, received, and processed. In the treatment of time in this paper, discrete time structure is assumed. Time is what a clock measures11. Creating time means the appearance of a time flowthing in the system, e.g., 9:00 AM is created by the employee attendance machine and flows to a database as a record of the time of an employee’s presence at work. The flowthing 9:00 AM is released and transferred by the attendance machine to be received and processed by the database module. Note that here the clock is the system clock synchronized to a “real time” clock. So, if the work shift in a factory is eight hours, then, in the timer subsphere of the company, the first hour (flowthing) is created, and then the second hour is created as a real-time hour after the first, etc. At the end of the shift, the timer signals the bell subsphere to ring, indicating the end of the shift. Here the timer plays the role of counter of hours.

Of course, time as a phenomenon is generated continuously. Bock12 identified five meanings in the context of UML requirements for continuity:

1. Mathematical properties, e.g., real numbers.
2. Continuous flow of items or control (UML 2 and SysML), e.g., oil flowing to a pump.
3. Continuously varying inputs or outputs, e.g., voltage at an input may vary as the sine of time.
4. Continuous execution (UML 2 and SysML), e.g., a car engine.
5. Continuous item (UML 2 and SysML), e.g., a tank of water or bucket of ball bearings.

In FM, time flows just like any other flowthing, as described above. A time flowthing can also be stored, destroyed, … but these are not exclusive actions like create, release, transfer, receive, and process. For example, flowthings can be stored while they are in the creation stage, or in the released stage, and so forth.

Example: Consider Fehnker’s13 example of an intelligent light switch (Fig. 4) that operates as follows:

- Press button twice quickly to switch to bright
- Press it once to switch to dimmed
- If light is on or dimmed, pressing button switches light off

Fig. 4 shows a representation of this system. It includes the spheres Clock, Clock control, Light control, Light, and User. All spheres except Light have a single flowsystem; hence, the sphere and the flowsystem are represented by one box. The Light has two flowsystems: Dim and Bright. Note that time, press, control signals, dim, and bright are treated as flowthings, e.g., dimness can be created and processed (less, more).
The clock (circle 1) creates and transfers (2) time slots. The Time control regulates their release to the Light control. They are either “destroyed” or passed. Time control plays the role of a time counter that passes a signal for each, say, second, to the Light controller. The signals can be thought of as clock ticks that are “heard” by the Light control.

When the user presses a switch once (circle 3), this triggers time control (4) in the Clock control. It is assumed that, initially, the Clock control blocks all clock signals. Thus, when user presses once, a number of time slots ($n$) flow (5) to Light control (6). If the user presses again, the Clock control is already ON (7) and this turns off sending the $n$ time slots to Light control. Accordingly, in Light control if the number of arriving time slots is equal to $n$ (8), this means the user has pressed once, and Dim is created (9). If Dim is already ON, it turns it OFF (10) along with the Bright light (11) if it was ON. If the number of time slots is less than $n$, this means the user has pressed twice and Bright light is created (12).

Accordingly, in human terms, the timing mechanism in FM is as illustrated in Fig. 6. Clock ticks arrive at a time controller who ignores them (a) until receiving an instruction (b) to divert them to a task controller who uses them to regulate tasks (c).

**Fig. 5. FM representation of intelligent light switch**

![Fig. 5. FM representation of intelligent light switch](image)

**Fig. 6. Illustration of timing mechanism in FM**

![Fig. 6. Illustration of timing mechanism in FM](image)
4.2 **Time as a sphere**

A flowthing in FM can be conceptualized as a sphere and vice versa. In UML, *timing diagrams* are used to show interactions “when a primary purpose of the diagram is to reason about time”\(^{14}\). Lifeline, state timeline, destruction event, duration constraint, and time constraint are typically drawn as nodes and edges in a timing diagram. Tomar\(^{14}\) gives a typical UML timing diagram as shown in Fig. 7. Fig 8 shows the diagram recast as a corresponding FM representation.

In Fig. 8, time intervals are drawn as spheres through which the flowthing Seminar flows. During the time *Nov 1… Dec 31*, a seminar is crafted (created) to “move” (flow) to the scheduling stage between *Jan 1 and July 31*, and so forth. The interesting aspect of this FM representation is that the same diagramming methodology is applied to timing description as for functional and other constraints specification. For example, the entrance of a seminar in the *August* period can trigger creation of a list of registered students, as shown in Fig. 9.

![Fig. 7. Timing diagram (from Tomar\(^{14}\))](image)

![Fig. 8. An FM timing diagram](image)

![Fig. 9. The same diagrammatic method is used to represent the flow of a seminar through time spheres and to represent other processes occurring during that period, e.g., for the month of August the seminar moves to the enrolment state, and this triggers activation of the student registration process.](image)

5. **Applying FM**

Consider the case study by Gezgin et al.\(^{15}\) of a car control system with a functional structure shown partially in Fig. 10. A sensor produces data used to extract line information indicating lanes in a street that control steering. Fig. 10 is incomplete and does not present a fair view of the involved project; however, it is sufficient for our purpose: to give a flavor of this type of high-level description of a real-time project.

Gezgin et al.\(^{15}\) use several types of diagramming methodologies; two of them are shown in Fig. 11. Their case study will be simplified in a related problem by assuming that the sensor produces deviation from a single straight line (Fig. 12).
Fig. 13 shows a portion of the FM representation of the control system for this system. It is assumed that:
- The sensor sends its data every 20 seconds
- If no data arrive at the controller, an emergency warning is generated.

Fig. 13. FM representation of a portion of a Car Control system
- If the arriving data are wrong (e.g., the sum of the left and right angles is not equal to 180), an emergency warning is generated.
- The controller corrects any deviation from the straight line in at most 15 seconds; otherwise, an emergency warning is generated.

In Fig. 13, when the car’s driving state is turned on (circle 1 in the figure), two things happen:
- The sensor is turned ON (2)
- Clock ticks (3) are triggered to count 20 seconds before generating the next sensor data. Upon reaching 20 seconds, a signal is created (4).

Accordingly, the turned-ON sensor sends its data (angle measurements) to the Control module (5) which indicates the arrival of data (6). If no data are received within 20 seconds (7), an emergency warning is created (8). Fig. 14 illustrates this type of error in the familiar join notation.

The control module also processes (10) the data and if incorrect, it creates: Emergency SENSOR DATA ERROR (11). If there is a deviation (12), then,
- A correction measurement is created (13) and sent to the actuator (14) which triggers (15) the action to correct the steering (6). Finishing such a task triggers creating a “correction done” signal (17).
- A 15-second counter is triggered to accept Clock ticks (18) and to create a signal (19) indicating the passing of 15 seconds.

Accordingly, the two signals “done” and “15 seconds” flow (20), and if “done” does not arrive, an Emergency TIME OUT FOR CORRECTION is created (21).

4. Conclusion

This paper introduces an alternative (e.g., to UML) diagrammatic description timing plan for systems. The usability and feasibility of the proposed method is illustrated by applying it to examples. The same diagramming methodology is applied for timing description as with functional and other constraints specification (e.g., data error in Fig. 13). The resultant description seems a promising approach as a very high-level system specification tool. Further work would explore additional characteristics of the FM diagrammatic methodology with regard to modeling of timing. Experimentation to verify the expected timing behavior could be conducted using simulation\textsuperscript{16,17}.

References


