Specifying Reactive Systems through the Object-Process Methodology

Mor Peleg and Dov Dori
Faculty of Industrial Engineering and Management
Technion¾ Israel Institute of Technology
Haifa 32000, Israel
{mor, dori}@ie.technion.ac.il

Abstract

There is general consensus in the software and control systems literature that real-time systems are difficult to model, specify, and design. This is due to the fact that issues such as concurrency, synchronization between processes, and real-time constraints must be expressed. Since real time systems play a central role in many technological advents, it is an important and challenging task to develop an intuitive and easy-to-use, yet coherent and concise method for specifying such systems. The Object-Process Methodology (OPM) graphically specifies systems in a single unified model that describes the static-structural and behavioral-procedural aspects of a system. OPM/T is an extension of OPM for specification of reactive and real-time systems, which is capable of expressing temporal constraints, referencing events, and handling exceptions. In this paper, OPM/T is presented and demonstrated through the specification of a home heating system.

Keywords: analysis and design methodology, reactive and real-time systems specification.

1. Introduction

Reactive systems and real-time systems play an important role in many technological fields. The Real time aspect is critical in such systems as chemical processing, road-traffic, nuclear reactions, satellite control, and airplanes and missile navigation. There is general consensus in the software and control systems literature [1] that real-time systems are difficult to model, specify, and design. This is due to the fact that issues such as concurrency, synchronization among processes, and real-time constraints must be expressed explicitly and unambiguously. It is an important and challenging task to develop an intuitive and easy to use, yet comprehensive and concise method for specifying such systems.

Reactive systems [2] are systems which are event-driven, having to react to external and internal stimuli occurring at random times. The behavior of reactive systems cannot be specified by merely giving the inputs and outputs of the system. There is a need to represent the **control** component which determines the order and timing of processes. One customary way of expressing a system’s response to events is by applying a set of Event-Condition-Action (ECA) rules. These rules specify that a process (action) in a reactive system is executed when the triggering event occurs, provided the conditions guarding the process execution are fulfilled.

Real-time systems are reactive systems in which timing constraints may be **quantitative**. The constraints may be on the time between an event and the system’s response, on the execution time of a process, on the time the system stays in a specific state, etc.

This paper presents OPM/T — a real-time extension to the Object-Process Methodology (OPM) [3]. The paper is organized as follows. Section 2 briefly surveys specification methods for reactive and real-time systems. Section 3 introduces and demonstrates the basics of OPM and OPM/T. Section 4 describes the details of the Heating System case study, which is an example of a real-time system and shows the OPM/T specification of the Heating System. A discussion concludes the paper.

2. Specification Methods for Reactive and Real-Time Systems

Specification of reactive and real-time systems is currently done in a number of methods, which can be roughly divided into two groups [1]. The first group consists of graphical methods, while the second one is based on logics and algebras. Following is a comparison of the different specification methods. The methods are compared in terms of their ability to model real-time systems and verify the specification, and their usefulness as providers of a sound basis for design and implementation.

Graphical methods used for specification of real-time systems can be categorized as: (1) system structure, function, and behavior modeling methods; (2) methods that specify system behavior only; and (3) real-time design methods.

The first group of methods includes, among others, Unified Modeling Language (UML) [4], Object Oriented Analysis (OOA), Object Modeling Technique (OMT) [5], Object Oriented Software Engineering (OOSE), Object Life Cycles, and Object Oriented System Analysis (OSA) [6], reviewed in [7]. The methods belonging to this category describe a system from three different aspects: structural — objects and the relationships among them, using mainly ERDs—Entity-Relationship Diagrams of various forms, functional — the processes executed in the system, including their inputs and outputs, using mainly DFDs—Data Flow Diagrams, and behavioral — control of process execution and changes in an object's state, using methods based on Finite State Machines and their elaborations, such as Statecharts.

While these methods are easy to use, and their resulting specification can be understood by non-experts, they suffer from two main disadvantages. First, they do not fully support expression of temporal constraints and reference to events. Second, to specify a system, these methods use a combination of at least three models, each describing a different aspect of the system. Incompatibility problems, such as mismatches among names of objects and processes are more likely to occur when more than one model is used. Furthermore, the integration of the different models that describe different aspects of the system is seldom explicit. Hence, the difficult task of mental integration has to be done by the analyst and the audience to which the analysis is intended. Except for OSA, all system structure, function, and behavior modeling methods have no formal semantics and no support for formal verification. OSA [6] can support validation based on prototype execution of analysis application models. Some of the modeling methods discussed above are supported by CASE tools, which are supposed to facilitate the transition from specification to design, and from design into executable code.

Statecharts [2], Modecharts [8], and Petri Nets [9] are methods that specify system behavior only. Both Statecharts and Modecharts are based on Finite State Machines, but they are also capable of hierarchical and structural decomposition into sub-states, which may coexist in parallel or exhibit an exclusive-or (XOR) relationship. Statecharts have been extended in [13], so as to express quantitative timing constraints. STATEMATE [2], the graphical tool which implements Statecharts, has an automated simulation tool that allows the user to execute his/her model. Modecharts [8] are capable of expressing quantitative sporadic and periodic timing constraints. Their specification is done in Real Time Logic (RTL) [1]. Although both Statecharts and Modecharts are formal methods, formal verification is not yet supported for their resulting specifications.

Petri Nets [9] is a formal graphical language that can express concurrency, nondeterminism, and cause-and-effect relationships between events and states. Timed Petri Nets were developed to express temporal constraints [1]. Petri Nets are especially useful for performance evaluation of modeled systems. They can be formally checked for boundedness, safety, and freedom from deadlock. Invariants can be checked for small systems. But Petri Nets can overkill if the system is too simple, while if the system is too complex, timing can become obscured.

Jackson System Development (JSD) [10], and Sanden's Entity-Life Modeling [11] are real-time design methods that can also be used for specification. Both methods focus on the implementation domain. The graphical diagrams these methods yield present the different tasks performed by objects and the communication among them. In addition to the graphical diagrams, these methods also rely heavily on pseudocode, which is not graphically represented.

Logics and algebras used for specification of real-time systems are reviewed by Ostroff in [1]. The main advantages of using such methods in specification of systems are that any temporal property can be specified, and that the specification can be verified for correctness by mathematical methods. Nevertheless, the task of specifying a system in logic is very difficult, and the resulting specification is hard to follow and understand by non-experts. Mechanical theorem provers have failed to be of much help due to the inherent complexity of validity testing. There are no available tools yet for design and implementation of specifications given in logics and algebras.

3. The Object-Process Methodology (OPM) and OPM/T

The Object-Process Methodology (OPM) [3] incorporates the static-structural and behavioral-procedural aspects of a system into a **single** unifying model. OPM achieves this by treating both **objects** and **processes** as **things** (entities) which have equal status. OPM handles complex systems by using recursive seamless scaling. It is suitable both for system analysis and system design, and enables smooth transition between theses phases.

In OPM, objects are viewed as persistent entities interacting with each other through processes—transient entities that affect objects by changing their state. Object-Process Diagrams (OPDs) enable us to describe things and how they interact with each other. Things can be simple or compound. A compound thing is a thing which is a generalization of other things, or an aggregation of other things, or is characterized by other things. Objects may serve as enablers — instruments or intelligent agents, which are involved in a process without changing their state, or they may be affected (or generated or consumed) by a process. OPDs can depict both sequential and concurrent processes, and accommodate expressions of branching.

OPM/T is an extension of OPM for specification of reactive and real-time systems. The OPM/T functionality includes triggering events, guarding conditions, temporal constraints, timing exceptions. Triggering events can be explicitly represented in OPDs by adding information to the procedural link directed from the triggering object to the corresponding triggered process. The letter *e* (for "event") added within the circle of an enabling link (agent or instrument), or next to the arrowhead of an effect link, specifies the fact that the link is a triggering event. The event can also be explicitly specified in text (i.e. *e: switch pressed*) which is recorded along the corresponding procedural link.

Guarding conditions are specified in a manner similar to that of events. The letter *c* is recorded within the circle of an enabling link or next to the arrowhead of an effect link connecting the object state that serves as a condition guarding the execution of a process. Figure 1

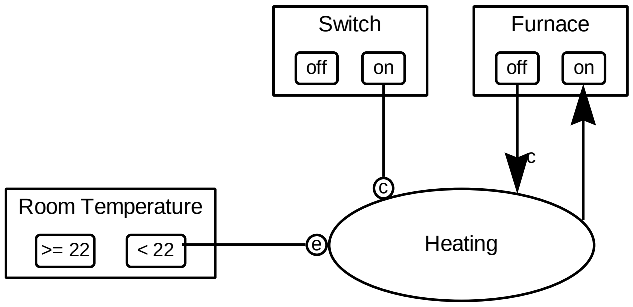

Figure 1: An OPD featuring the triggering event and guarding conditions of the *Heating* process.

shows an OPD featuring a triggering event and guarding conditions of a *Heating* process. It specifies the fact that on the event of the Room Temperature entering a state of "<22" degrees, the *Heating* process is triggered under the conditions that the Switch is in the *on* state, and the Furnace is in the *off* state. The *Heating* process changes the state of the Furnace to *on*.

Temporal constraints are expressed by specifying an interval "(x,y) ", where $0 \leq x \leq y$. x and y represent the lower and upper bounds of the interval, respectively. The case where $x = y$ represents an interval (x,y) of length zero, which denotes a point in time. y can also be ∞. In this case, the interval (x,∞) is bounded only by the lower bound, x. There are three kinds of temporal constraints:

- process duration constraint, in which the interval (x,y) is recorded inside the constrained process, as in figure 2a.
- state duration constraint, in which the interval (x,y) is recorded inside the constrained object state, as in fig. 2b.
- reaction time constraint, in which the interval (x,y) is recorded above the procedural link connecting the triggering object to its triggered process, as in fig. 2c.

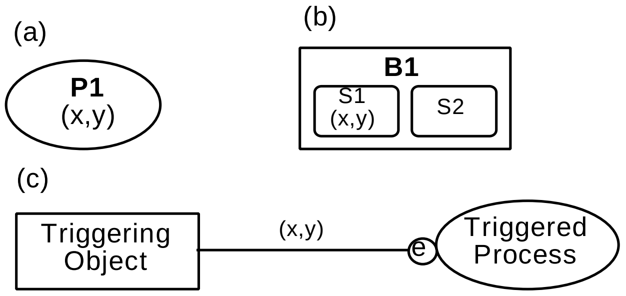

Figure 2: Expression of temporal constraints in OPDs. (a) process duration constraint; (b) state duration constraint; (c) reaction time constraint.

A *timing exception* is a violation of a temporal constraint. Certain exceptions are quite common and acceptable, while others may be rare, albeit hazardous. In case such an exception occurs, it is desirable to perform a suitable exception handling process. In order to represent exception triggers, the Exception Link, denoted as —|—, adopted from [6], is used to connect the violated constraint to the exception triggered process.

4. The Heating System and Its OPM/T Specification

The Heating System case study, adapted from [12], specifies a home heating system. By turning the furnace on and off, the controller of the home heating system regulates in-flow of heated water that circulates in the system and heats the home, provided that the master switch is set to *heat* position. The formal system requirement specification (SRS) document for our case study is listed below.

1. The controller starts monitoring the room temperature within 1 second after the main switch is turned to heat.
2. The controller sends a signal to activate the motor 1 second after the monitored room temperature has fallen below $(H_d - 2)$ degrees, where H_d is the desired temperature set by the user.

3. The controller monitors the motor speed. Within 1 second after the speed reaches a predefined value, S_d, the controller signals the oil valve to open and ignites the furnace.
4. The controller also monitors the water temperature. Within 1 second after the temperature has reached a predefined value, T_w, the controller signals the circulation valve to open. The heated water then starts to circulate through the house and heat it.
5. A fuel level indicator and a combustion sensor send exception signals to the controller if abnormalities occur, in which case the controller signals the furnace to shut off within 0.1 seconds.
6. Once the home temperature reaches (H_d+2) degrees, the controller deactivates the furnace first by closing the oil valve within 1 second, and 5 seconds later, by closing the circulation valve and stopping the motor.

The system must also abide by the following constraints:

- The minimum time interval between the Furnace's Motor restart and the end of the most recent operation is 5 minutes (300 seconds).
- The Furnace cannot run for more than *max* consecutive seconds.

Figure 3 is a top-level OPD of the Heating System. The Heating System object consists of three parts: Switch, Monitoring System and Furnace, and is characterized by two objects: Monitored Value-set and Predefined Constant-set. The Environment object consists of two parts: User and Room Temp. At the same top-level OPD, four processes are also depicted: *Switching* (*P1*), *Room Temperature Monitoring* (*P2*), *Heating* (*P3*), and *Furnace Shut-off* (*P4*). Processes and objects with a bold contour are zoomed in (scaled up) in other, lower level, OPDs. The background shade of scaled-up things is identical to their shade in their non-scaled-up version.

4.1 Triggering events and exception links

Each process is triggered by one of its triggering events. These events may be external events, events that mark the change in an object's state, events that mark the termination of a (pervious) process, and exception events.

External events and events that mark a change in an object's state are specified by the letter *e* at the process end of the relevant procedural links (effect, agent, and instrument links) emanating from the triggering object to the triggered process. For example, in the OPD of Figure 3, *Switching* (*P1*) is triggered by the external event *e:* Switch pressed, and *Room Temperature Monitoring* (*P2*) is triggered by the event of Switch entering the *heat* state.

A process can also be triggered by an event that marks the termination of some process. This triggering is done via the invocation link. In the case of *P2*, whose details are given in Figure 4, the invocation link is reflexive—it loops in and out of *P2*, meaning that the event of termination of *P2* invokes another iteration of that process, and this iteration keeps recurring as long as the condition requiring that the state of Switch is *heat* holds.

Another method of process triggering is via the *exception link*. An exception link connects the violated timing constraint to the exception handling triggered process. In the OPD of Figure 3, *Furnace Shut-off* (*P4*) can be triggered by one of four events. One of these events is a timing violation of the state duration constraint, imposed upon the *on* state of Furnace, restricting the duration of the time spent in that state to be within the interval (0, max). This violation is specified by the exception link, which activates the process *P4*. Note that the default logical relation among two or more triggering events of a single process is OR, meaning that any one of them alone can trigger the process. Thus, for example, the occurrence of any one of the triggering events of *Furnace Shut-off* is sufficient to shut Furnace off. The default logical relation between guarding conditions, and between objects which participate in a process (as enablers or as affected objects) is AND, meaning that they are all required for the process to occur. In order to denote non default relations between objects that participate or guard a process, the XOR and OR links can be used. Such links can also be used in order to specify that different guarding conditions apply depending on which event triggers the process. A compound triggering event is the composition of two or more simple (atomic) events, which may occur in any specified order and timing constraints. Such an event is specified by a fork, which connects the atomic events into one compound event. An example of such a triggering event is shown Figure 6, in which process *P4.2* is triggered by the compound event, in which both Oil Valve enters the *closed* state, and Furnace enters the *off* state.

4.2 Temporal constraints

As noted, temporal constraints are classified into process duration, state duration, and reaction time constraints. In the example given above, the *on* state of Furnace has a duration constraint attached to it, which restricts the time spent at that state to be within the interval (0, max). Another state duration constraint appears in Figure 5, in which the *off* state of Motor is constrained by the interval (300,∞), specifying that once

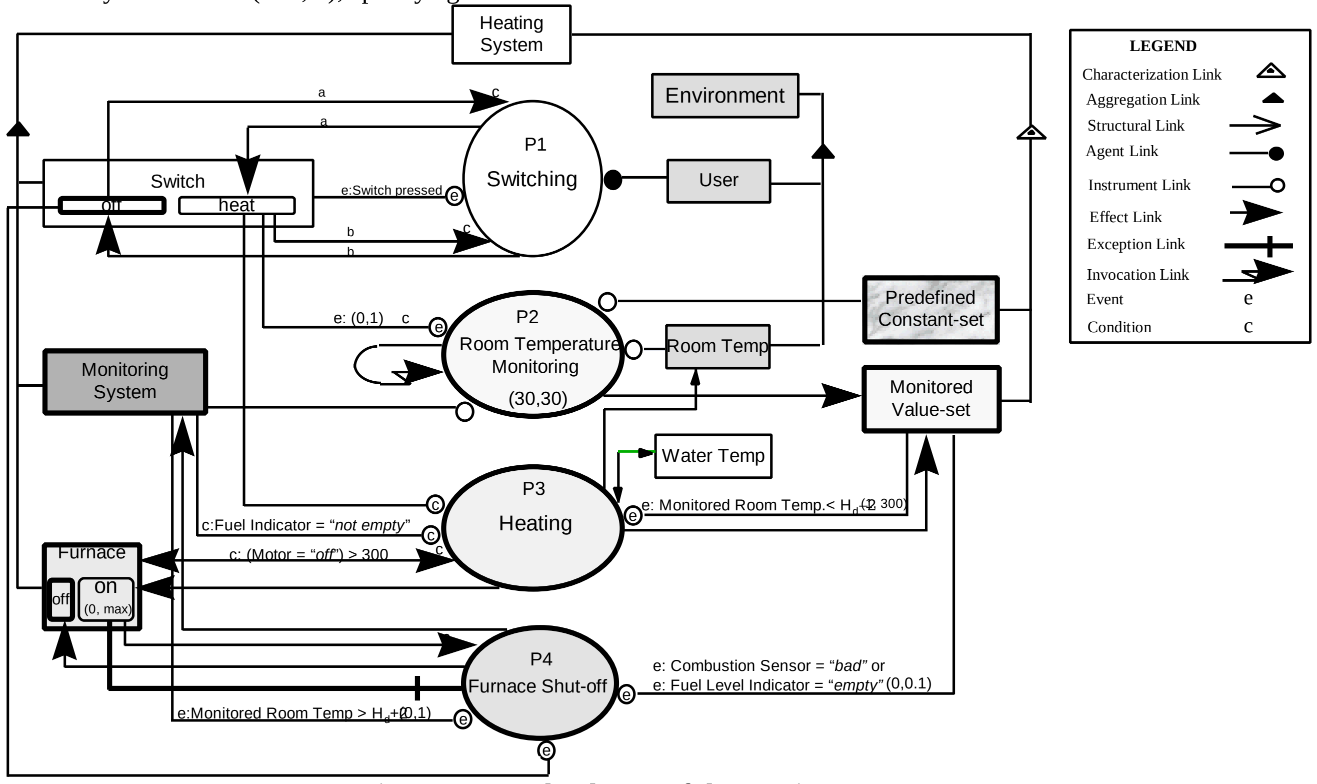

Figure 3: A top-level OPD of the Heating System

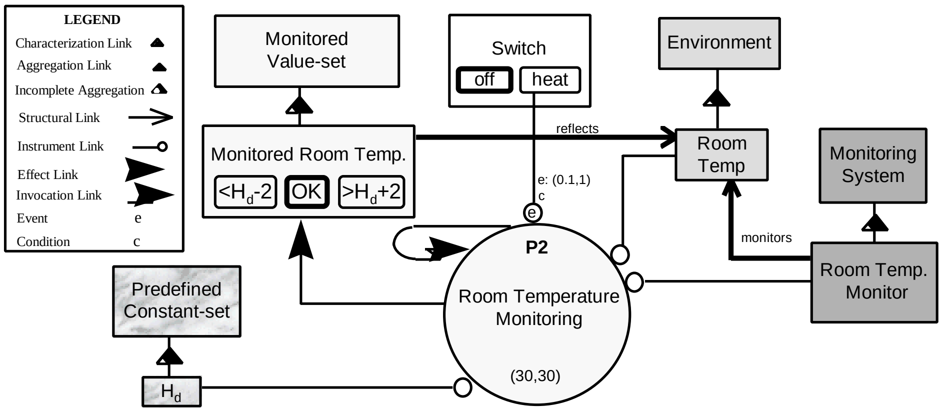

Figure 4: An OPD showing details of *P2: Room Temperature Monitoring*

the motor has been turned off, it must remain in that state for at least 300 seconds.

An example of a process duration constraint is the (30,30) interval that restricts the duration of each iteration of the reflective process *Room Temperature Monitoring* (*P2*), shown in Figures 3 and 4, to be exactly 30 seconds long. Together with the invocation link, which loops out and in of *P2*, it specifies that this process is executed repeatedly, with a period of 30 seconds. Reaction time constraints, many of which appear in the OPD set of the heating system, are recorded above procedural links emanating form triggering objects to their triggered processes.

The reaction time constraint (0,1), shown in Figure 3, for example, is imposed upon the time passing from the event of Switch entering the *heat* state to the event marking the beginning of *P2*. Another example for a reaction time constraint appears in Figure 5, in which (1, 300) is the reaction time constraint imposed upon the instrument link connecting the Monitored Room Temp.

“< *(H_d- 2)* ” state to the process *Furnace Motor Turn On* (*P3.1*). The triggering event of *P3.1* occurs when

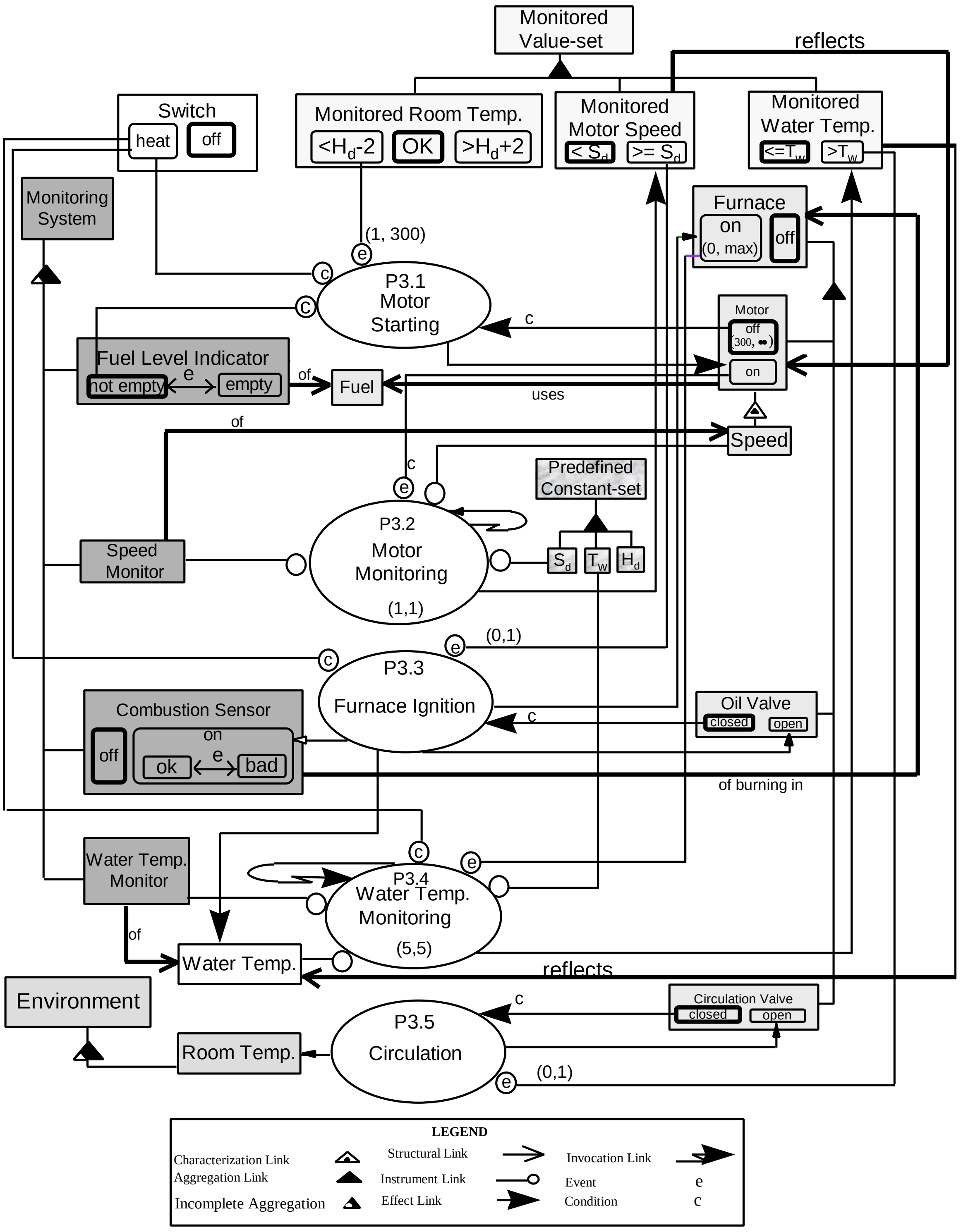

Figure 5: An OPD showing the explosion of *P3: Heating*

Monitored Room Temp. achieves a value lower than (H_d- *2*) degrees. The process *P3.1* is triggered 1 second after the above triggering event has occurred, provided that (1) Switch is in the *on* state, (2) Fuel Indicator is in the *not empty* state, and (3) Motor has been in the *off* state for at least 300 seconds. If Motor has been in the *off* state for (t<300) seconds when the triggering event — Monitored Room Temp. falling below (H_d- 2) degrees —

occurs, then process *P3.1* occurs after (300- *t*) seconds, i.e., as soon as the time spent in Motor's *off* state reaches 300 seconds. This is achieved since the reaction time constraint of the triggering event allows the reaction time to be in the interval (1, 300) seconds, while concurrently Motor is constrained to be in the *off* state for at least 300 seconds, as denoted by the (300,∞) state duration constraint of that state.

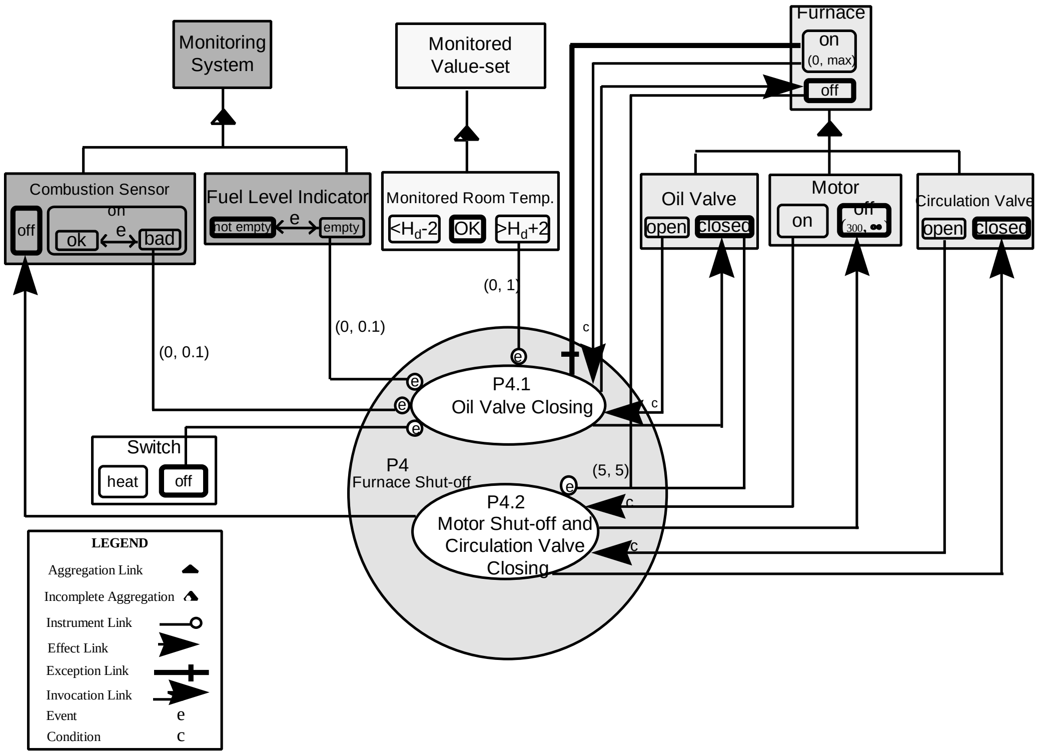

Figure 6: An OPD showing the blow-up of *P4: Furnace Shut-off*

OPM/T helps clarifying and communicating the structure and behavior of the entire system in the following ways:

1) Following the default scenario (normal sequence of processes and events) is easy, since it is reflected by the top-to-bottom order in which the processes in OPDs are depicted.

2) For each process, all relevant information is readily visible. This includes the triggering events, guarding conditions, timing constraints, all the affected or consumed objects which serve as the process inputs, the affected or generated output objects and the enablers—objects that participate in the process without changing their state. State changes of the affected objects that result from the process execution are also shown explicitly.

3) The system's static-structural and functional-behavioral aspects are presented from a top-level view, and zoomed in (scaled up) to provide more and more details. Scaling makes it possible to depict only those objects that participate in the processes shown in an OPD. These objects are scaled up to the level that exposes all the relevant parts and/or specializations and/or features (attributes and methods) of each object. This way, each OPD does not contain too many links, which can make the diagram cluttered. Scaling also enables the system analyst/designer to present different views of the same system, as desired. If convenient, scaling can be used to construct the three separate views of the structural, functional, and behavioral aspects of a system, as is done by other visual specification methods. The connections between the different OPDs that make up the OPD-set of a given system under consideration are maintained through the scaling mechanism; Each process or object in a given OPD from an OPD-set can be scaled-up to show more details in another OPD from the same OPD-set. Two or more things in a given OPD from an OPD-set can be scaled-down to hide details, in another OPD from the same OPD-set.

4) Processes in OPDs group together all objects which are transformed by the same event (which may be simple or compound). This grouping is also useful for

expressing synchronization of the different reactions that trigger the process and/or are triggered by it.

Discussion

The complexity of real life systems in general and those having a real-time element in particular requires a method that can describe them in a coherent and consistent manner. Object-oriented methods advocate the use of a combination of several models – usually a static-structural model, a functional model, a behavioral model, and a host of additional models – to describe the various aspects of the system being analyzed. The use of several models to describe a system from various aspects is the cause for the *model multiplicity problem*, in which the various models have to be referred to concurrently in order to get a real, deep comprehension of the system and the way it operates and changes over time. Real-time requirements complicate this problems even further, as either yet another model needs to be introduced to reflect these specifications or they are added into one of the existing models (usually the behavioral model).

When we consider a complex system that features an involved dynamic behavior we think of the structure and the behavior of components of the system in tandem – it would be most unnatural to complete constructing the system's structure, then put it aside and deal exclusively with the system's behavior. Yet, this is exactly what object-oriented analysis and design approaches require that we do.

The Object-Process Methodology (OPM) has been developed to cater to the natural train of thought that a human normally applies while trying to understand complex systems. In such systems, it is often the case that structure and behavior are intertwined so tightly that any separation between them is bound to complicate further the already complex description. OPM/T is the real-time extension of OPM. It is designed to express time-dependent elements such as triggering events, guarding conditions, timing constraints, timing exceptions, and flow of control constructs.

We have used the Heating System case study to demonstrate how a real-time system can be easily specified and understood in OPM/T via a single set of Object-Process Diagrams (OPD set) enhanced with the appropriate time elements. The fact that the model is single makes it easier to understand the system as a whole, as it enables one to grasp the system's structure, function, and behavior by studying an integrated, coherent view of the system. Furthermore, working within a single model, it is less likely to generate and encounter incompatibilities in objects, processes, and events.

OPM and OPM/T are especially suitable for specifying systems which contain many objects and/or processes, and in particular if they need to be specified at different detail levels which are taken care of by OPM's scaling option. Refinement of OPM/T, including identification, characterization, and classification of real-time design patterns is currently under way. The implementation of the real-time extensions is being incorporated into OPCAT—Object Process CASE Tool, which is currently under development.

References

1. Ostroff, J.S., Formal Methods for the Specification and Design of Real-Time Safety Critical Systems. *The Journal of Systems and Software* Vol 18 No 1, 33-60 (April 1992).
2. Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtull-Trauring, A., and Trakhtenbrot, M., STATEMATE: A Working Environment for the Development of Complex Systems *IEEE Transactions on Software Engineering* Vol 16 No 4, 403-414 (April 1990).
3. Dori, D., Object-Process Analysis: Maintaining the Balance Between System Structure and Behavior *Journal of Logic and Computation* Vol 5 No 2, 227-249 (April 1995).
4. Booch, G., Jacobson, I., and Rumbaugh, J., *Unified Modeling Language (UML) Notation Guide Version 1.1*, Rational Software Corporation, September 1 1997.
5. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorenson, W., *Object-Oriented Modeling and Design* Prentice-Hall Englewood Cliffs NJ, 1991.
6. Embley, D.W., Jackson, R.B. and Woodfield, S.N., Object-Oriented Systems Analysis: Is It or Isn't It? *IEEE Software* Vol 12 No 4, 19-33 (July 1995).
7. Dori, D. and Goodman, M., On Bridging the Analysis-Design and Structure-Behavior Grand Canyons with Object Paradigms *Report on Object Analysis and Design* Vol 2 No 5, 25-35 (January-February 1996).
8. Jahanian, F. and Mok, A.K., Modechart: A specification Language for Real-Time Systems *IEEE Transactions on Software Engineering* Vol 12 No 12, 933-947 (December 1994).
9. Peterson, J.L., *Petri Net Theory and the Modeling of Systems* Prentice-Hall Englewood Cliffs NJ, 1981.
10. Jackson, M., *System Development* Prentice-Hall International, 1983.
11. Sanden, B., Entity-Life Modeling and Structured Analysis in Real-Time Software Design A Comparison *Communications of the ACM* Vol 32 No 12, 1458-1466 (December 1989).
12. Toetenel, H. and Katwijik, J.V., Stepwise Development of Model-Oriented Real-Time Specifications From Action/Event Models *Lecture Notes in Computer Science* Vol 571 Springer-Verlag, 1991, pp. 547-570.
13. Drusinsky, D. and Harel, D., Using Statecharts for Hardware Description and Synthesis *IEEE Transactions on Computer-Aided Design* Vol 8 No 7, 798-807 (July 1989).