Towards Integrated System and Software Modeling for Embedded Systems

Hassan Gomaa

Department of Computer Science
George Mason University, Fairfax, VA
hgomaa@gmu.edu

Abstract. This paper addresses the integration of system modeling and software modeling, particularly for embedded systems, which are software intensive systems that consist of both hardware and software components. This paper describes a systems modeling approach to create structural and behavioral models of the total system using SysML. The systematic transition to software modeling using UML is then described.

Keywords: systems modeling, software modeling, SysML, UML, embedded systems, structural modeling, behavioral modeling.

1 Introduction

Model-based systems engineering [1, 2] and model-based software engineering [3, 4, 5, 6] are increasingly recognized as important engineering disciplines in which the system under development is modeled and analyzed prior to implementation. In particular, embedded systems, which are software intensive systems consisting of both hardware and software components, benefit considerably from a combined approach that uses both system and software modeling. This paper describes a modeling solution to this problem with an approach that integrates these two disciplines for the development of embedded systems. In particular this paper concentrates on developing the hardware/software boundary of a system, the decomposition of the system into hardware and software components, and designing the interface between hardware and software components. The modeling languages used in this paper are SysML [7] for systems modeling and UML [8, 9] for software modeling.

This paper describes a systems modeling approach to develop a multi-view model of the system, in terms of a structural block definition diagram of the problem domain, a system context block definition diagram, a use case model, and a state machine model, which forms the basis for a transition to software models. This is followed by modeling the hardware/software boundary, which involves decomposing the system into hardware and software components and modeling the possible deployment of components. The steps in software modeling that address the software side of the hardware/software interface are described next, in which the boundary of the software system is established and the software components are determined. From

there, the software components are categorized as active (i.e., concurrent) or passive and their behavioral characteristics are determined. Although the approach can be used for most systems, it is intended in particular for embedded systems, which are software intensive systems that typically have a large number of hardware components, including sensors and actuators, which need corresponding software components to interface and communicate with them. An example of this model-based approach is given using a Microwave Oven system.

2 System and Software Modeling

The objective of the model-based approach described in this paper is to clearly delineate between systems modeling and software modeling, with a well-defined transition between the two phases. This section describes the steps in systems modeling, hardware/software boundary modeling, and software modeling.

2.1. Overview of System Modeling

System modeling consists of structural and behavioral (dynamic) modeling of the total system using SysML to get a better understanding of the system. The following steps consider the total system perspective, consisting of hardware, software and people, without consideration of what functionality is carried out in hardware and what functionality in software.

1. Structural modeling of the problem domain using block definition diagrams. In structural modeling of the problem domain, the emphasis is on modeling real-world entities, including relevant systems, users, physical entities and information entities.
2. System context modeling. A system context block definition diagram explicitly shows the boundary between the total system, which is treated as a black box, and the external environment. In considering the total hardware/software system, users and external systems are external to the system, while hardware and software entities are inside the system.
3. Use case modeling. In order to get an understanding of the system behavior, use case modeling is carried out. This involves determining the actors (users) of the system and the sequence of interactions between the actor(s) and the system.
4. Dynamic state machine modeling. State machines provide a more precise method for modeling the behavior of state-dependent embedded systems. For these systems, state machine modeling is preferred to activity diagrams because embedded systems are highly state dependent.

2.2. Overview of Hardware/Software Boundary Modeling

The following steps address the decomposition of the total system into hardware and software components, in particular determining what is done by hardware and what is done by software.

1. Decompose system into hardware and software components. Develop a block definition diagram that depicts the hardware components and the software system.
2. Deployment modeling. Develop deployment diagram depicting the deployment of hardware and software components.

2.3 Overview of Software Modeling

Once the hardware/software boundary has been determined, the next steps address the decomposition of the software system into its constituent components. They involve determining the boundary of the software system, and determining those software components that interface to and communicate with the hardware components.

1. Software context modeling. Unlike system context modeling, software context modeling depicts the boundary of the software system with the hardware components external to the software system.
2. Software component structuring. This step involves determining the software components that are needed to interface to and communicate with the hardware components. The software components are further categorized as active (concurrent) or passive.
3. Having determined the boundary between the hardware and software components, the subsequent development steps follow a UML-based software modeling and design method, in particular the COMET [4] method.

3. System Modeling

3.1 Structural Modeling of the Problem Domain

In structural modeling of the problem domain, the initial emphasis is on modeling real-world entities, including relevant systems, users, physical entities and information entities. Physical entities have physical characteristics – that is, they can be seen and touched. Such entities include physical devices, which are often part of the problem domain in embedded applications. Information entities are conceptual data-intensive entities that are often persistent – that is, long-living. Information entities are particularly prevalent in information systems (e.g., in a banking application, examples include accounts and transactions). Structural modeling using block definition diagrams allows the representation of these real-world entities as

blocks, as well as the relationships among these blocks, in particular associations, whole/part (composition or aggregation) relationships, and generalization/specialization relationships. Composite relationships are used to show how a real-world system of interest is composed of blocks. In embedded systems, in which there are several physical devices such as sensors and actuators, block definition diagrams can help with modeling these real-world devices. Individual entities are categorized as input devices, output devices, timers and systems, and depicted on block diagrams using stereotypes.

As an example, consider the structural model of the problem domain for the Microwave Oven System, which is an embedded system and is depicted on the block definition diagram in Fig. 1. The Microwave Oven System is modeled as a composite component, which senses and controls several I/O devices through sensors and actuators respectively. The oven is composed of three input devices: a door sensor which senses when the door is opened and closed by the user, a weight sensor to weigh food, and a keypad for entering commands. There are two output devices: a heating element for cooking food and a display for displaying information and prompts to the user. There is also a timer component, namely the real-time clock.

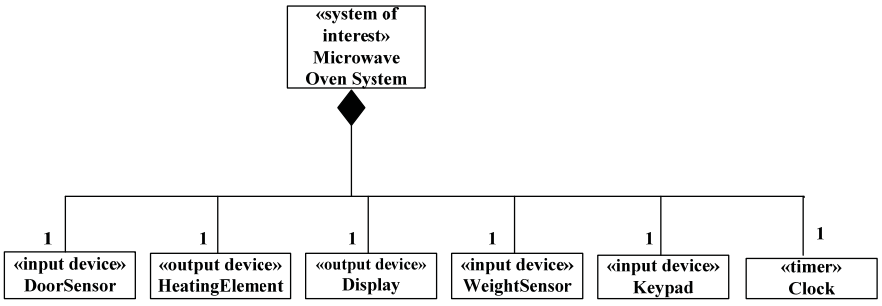

Fig. 1 Block definition diagram for Microwave Oven System

3.2 Structural Modeling of the System Context

It is very important to understand the scope of a computer system – in particular, what is to be included inside the system and what is to be excluded from the system. Context modeling explicitly identifies what is inside the system and what is outside. Context modeling can be done at the total system (hardware and software) level or at the software system (software only) level. A system context diagram explicitly shows the boundary between the system (hardware and software), which is treated as a black box, and the external environment. A software system context diagram explicitly shows the boundary between the software system, also treated as a black box, and the external environment, which now includes the hardware.

In developing the system context (which is depicted on a block definition diagram) it is necessary to consider the context of the total hardware/software system before

considering the context of the software system. In considering the total hardware/software system, only users and external systems are outside the system, while hardware and software components are internal to the system. Thus, I/O devices are part of the hardware of the system and therefore appear inside the total system.

To differentiate between different kinds of external entities, stereotypes [4, 9] are used. For the system context diagram, an external entity could be an «external system», when the system to be developed interfaces to either a previously developed system or a system to be developed by a different organization, or an «external user», to model a user of the system.

As an example, consider the total hardware/software system for the Microwave System. The system context diagram (shown on the block definition diagram in Fig. 2) is modeled from the perspective of the system to be developed, which is the Microwave Oven System and is categorized as «system». External to the system is the external user (modeled as an actor, see below) who uses the oven to cook food.

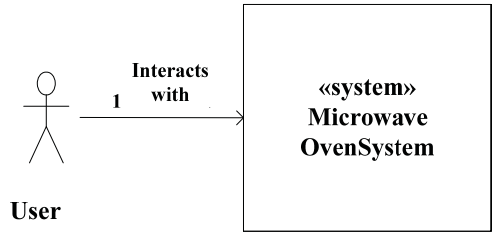

Fig. 2 System context diagram for Microwave Oven System

3.3 Use Case Modeling

In order to get an understanding of the system behavior, use case modeling [3, 4] is carried out. A use case describes a sequence of interactions between an actor (which is external to the system) and the system. In information systems, actors are usually humans. However, for embedded systems, actors could be also be external systems. In addition, a primary actor initiates the sequence of use case interactions. It is also possible to have one or more secondary actors that participate in the use case.

For the Microwave Oven System, the only actor is the user. In this simple example, there is one use case, Cook Food (see Fig. 3). The use case describes a main sequence in which the actor opens the door, places the food in the oven, closes the door, enters the cooking time, and presses the start button. The oven starts cooking the food and sets the timer. When the timer expires, the oven stops cooking the food. There are alternative to the main sequence, which the system has to be capable of handling, such as user opening the door before the food is cooked or pressing start before the time has been entered.

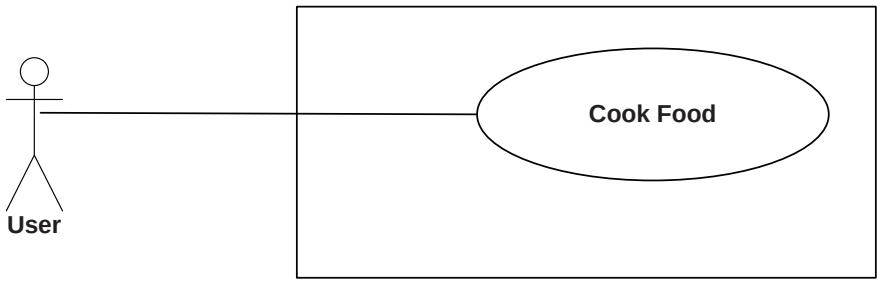

Fig. 3 Use case diagram for Microwave Oven System

3.4 Dynamic State Machine Modeling

Use case modeling provides an informal textual statement of requirements, which is usually sufficient for information systems but is unlikely to be precise enough for state-dependent embedded systems. State machines provide a more precise method for modeling such systems. A state machine can be developed by starting from the use case description and carefully considering all possible state-dependent scenarios. In particular, many events on the statechart correspond to inputs from the external environment, such as opening the door and placing the item in the oven, and many actions are outputs to the external environment, such as starting and stopping cooking.

The dynamic behavior of the microwave oven can be modeled as a state machine and depicted on a state machine diagram (also referred to as statechart [10]) as shown on Fig. 4, which depicts the states, events, and actions. The scenario described in the main sequence of the use case involves transitioning through the following states: Door Shut, Door Open, Door Open with Item (when item placed), Door Shut with Item, Ready to Cook (when cooking time entered), Cooking (when Start is pressed), Door Shut with Item (when timer expired), Door Open with Item, Door Open, and finally Door Shut. Many other transitions are possible corresponding to alternative scenarios.

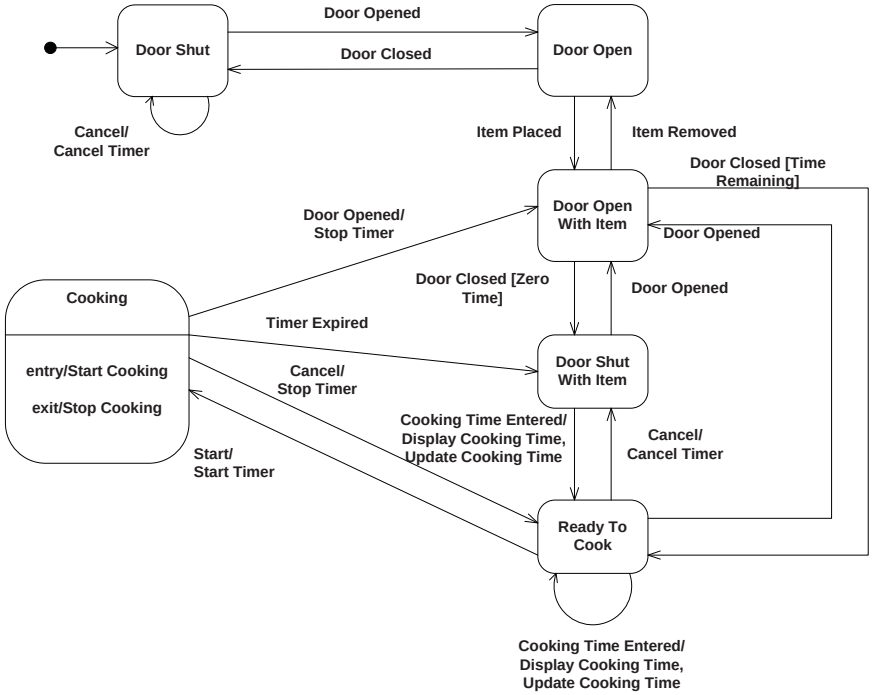

Fig. 4 State machine diagram for Microwave Oven System

4. Hardware/Software Boundary Modeling

Given the system requirements in terms of the structural model of the problem domain, the system context diagram, the use case model, and the state machine model, the system modeler can now start considering the decomposition of the system into hardware and software components. Hardware and software components are categorized using UML stereotypes.

4.1 Modeling System Decomposition into Hardware and Software Components

To determine the boundary between the hardware and software components, the modeler starts with the structural model of the problem domain (Section 3.1 and Figure 1) and then determines the decomposition into hardware and software components. The physical hardware components are explicitly modeled on the hardware/software block diagram while the software system is modeled as one composite component. In particular, the physical entities of the problem domain, as described in Section 3, are often input and/or output hardware devices that interface to the software system.

An example of a hardware/software system block diagram is given for the Microwave Oven System in Fig. 5. For this system, the devices originally identified in the structural model of the problem domain are analyzed further. The six part components of the Microwave Oven System identified in Figure 1 all have a hardware component to them, which are the hardware sensors and actuators that interface to the software system. There are three hardware input device components, Door Sensor, Weight Sensor, and Keypad (which all provide inputs to the Microwave Oven Software System), and two hardware output device components, Heating Element and Display (which receive outputs from the Microwave Oven Software System). There is also a real-time Clock hardware timer component, which signals the Microwave Oven Software System. The hardware components are categorized using UML stereotypes.

4.2 Deployment Modeling

The next step is to consider the physical deployment of the hardware and software components to hardware and software platforms. One possible configuration for the Microwave Oven System is depicted in the UML deployment diagram in Figure 6, in which the hardware and software components are deployed to different nodes physically connected by means of a high speed bus.

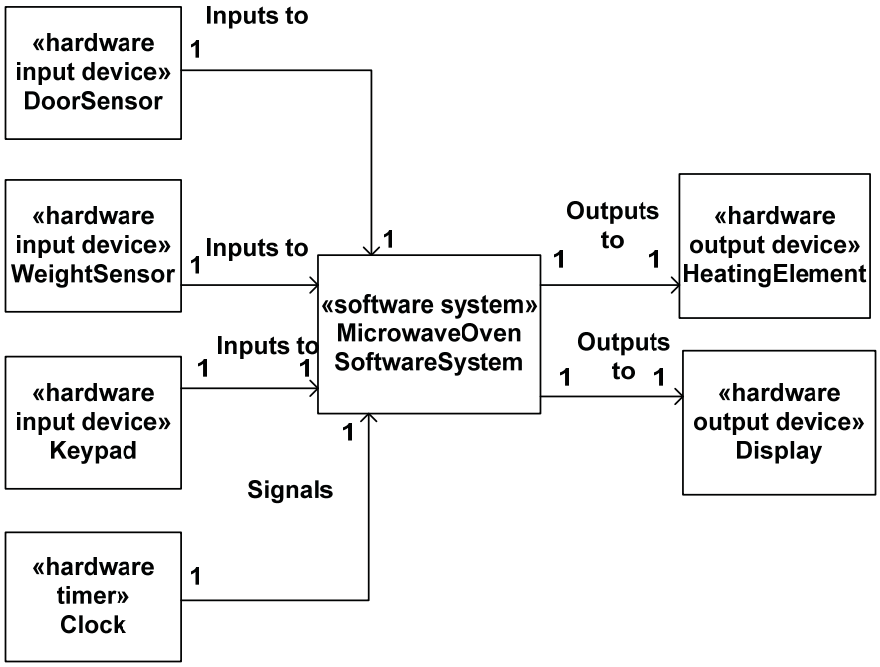

Fig. 5 Hardware/Software block diagram for Microwave Oven System

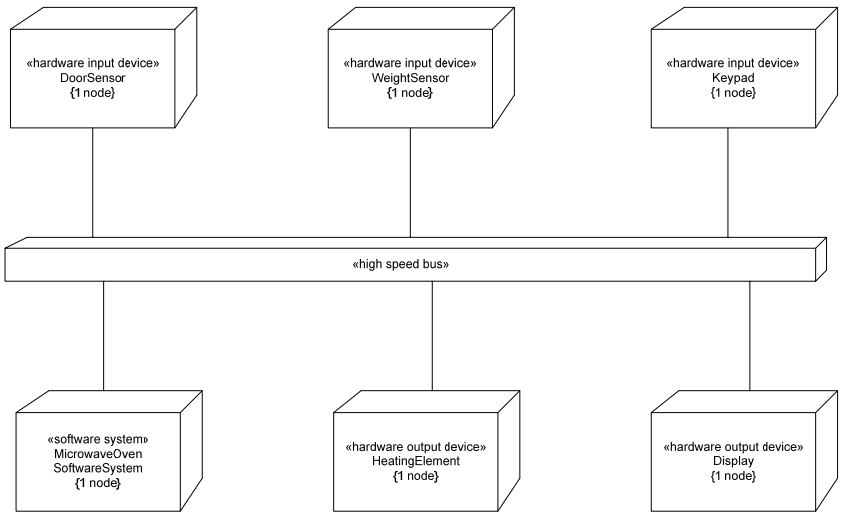

Fig. 6 Deployment diagram for Microwave Oven System

5. Software Modeling

The system context diagram depicts the systems and users that are external to the total system, which is modeled as one composite component. The hardware and software components are internal to the system and are therefore not depicted on the system context diagram. Together with the decomposition of the system into hardware and software components, this is the starting point for the software modeling.

5.1. Software System Context Modeling

The software system context model, which is used to model the boundary of the software system, is depicted on a class diagram and is determined by structural modeling of the external components that connect to the system. In particular, the physical hardware devices modeled on the hardware/software diagram are external to the software system.

The software system context diagram is modeled from the perspective of the software system to be developed, the Microwave Oven Software System, as shown in Fig. 7. From the software system point of view, the hardware sensors and actuators are external to the software system and interface to and communicate with the software system. Thus the hardware devices are external input and output devices, and an external timer as depicted in Figure 7, which is structurally similar to Fig. 5. However the categorization of the stereotypes is from the software system's point of

view. Thus the hardware devices on Fig. 5 are categorized as external devices on Fig. 7.

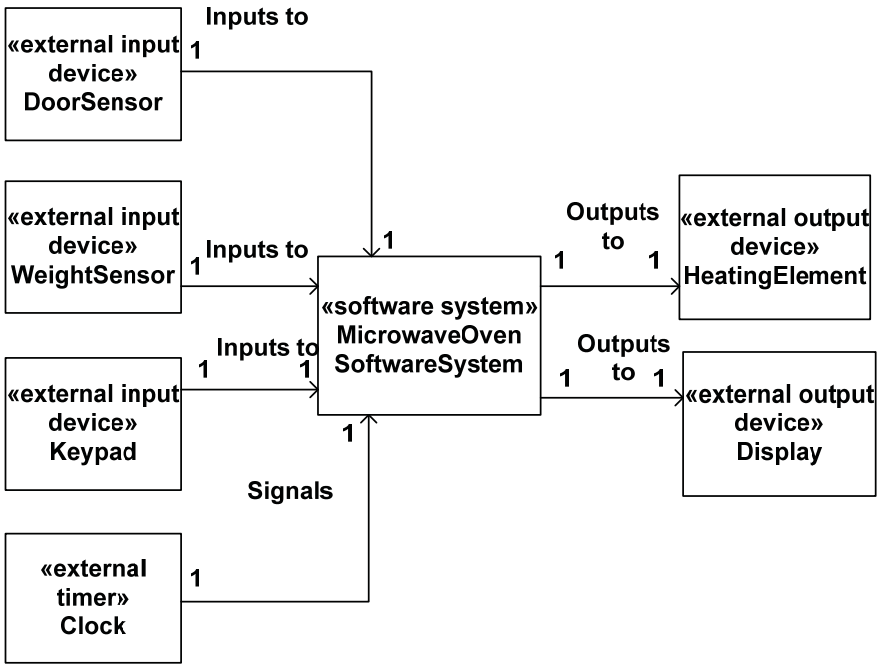

Fig. 7 Software context diagram for Microwave Oven Software System

5.2. Software Component Structuring

The next step is to determine the software components starting from the software system context model and working from the outside (hardware components) inwards to the software boundary components, which interface to and communicate with the hardware components. For every hardware component, there is a corresponding software boundary component, which is categorized using a UML stereotype. To receive input from an external input device, there needs to be a software input component. Each external output device component needs to receive output from a software output component. Each external hardware timer needs to signal a software timer component.

Software component structuring for the Microwave System is depicted on the class diagram in Fig. 8. Every external hardware component on the software system context diagram has a corresponding internal software component. Thus, there are three input

software components, Door Sensor Interface, Weight Sensor Interface, and Keypad Interface, which receive inputs respectively from the Door Sensor, Weight Sensor, and Keypad external input devices. There are also two output software components, Heating Element Interface and Display Interface, which output to the Heating Element and Display external output devices, respectively. There are two additional components, a state-dependent control component, Microwave Oven Control, which executes the microwave oven state machine depicted in Figure 4, and an entity component, Oven Data, which contains data about the cooking time. In addition, there is one timer component, Microwave Timer, which receives periodic inputs from the hardware Clock.

5.3. Software Concurrent Task Design

A characteristic of real-time embedded systems is that of concurrent processing in which many activities occur simultaneously and the order of incoming events is frequently unpredictable [11]. Consequently, it is desirable for a real-time embedded system to be structured into concurrent tasks (also known as concurrent processes or threads).

During concurrent task design, the system is structured into concurrent tasks and the task interfaces are defined [4, 11]. As before, stereotypes are used to depict the different kinds of tasks. Each task is depicted with two stereotypes, the first is the role criterion, such as input or control. The second stereotype is used to depict the type of concurrency.

Thus, an active «I/O» component is concurrent and is categorized further using a second stereotype as one of the following: an «event driven» task, a «periodic» task, or a «demand» driven task. Stereotypes are also used to depict the kinds of devices to which the concurrent tasks interface. Thus, an «external input device» is further classified, depending on its characteristics, into an «event driven» external input device or a «periodic» external input device.

Figure 8 is the starting point for designing the concurrent tasks, which are depicted using the UML 2 notation of parallel lines on the left and right hand side of the object box, as depicted in Figure 9. The three input software components, Door Sensor Interface, Weight Sensor Interface, and Keypad Interface, are designed as event driven tasks, since they are awakened by interrupts from the respective input devices (see below). The two output software components, Heating Element Interface and Display Interface, are designed as demand tasks, since they are awakened by the arrival of messages from Microwave Control. Oven Timer is a periodic task since it is awakened by the arrival of timer events from the external clock. Microwave Oven Control is designed as a demand task, since it is awakened by messages from the input tasks or the periodic task.

The entity objects Oven Data and Display Prompts are passive objects and do not have a thread of control. Because the entity objects are passive, they cannot be deployed independently of other components. Furthermore the passive objects are composed into a composite component with the tasks that access the passive objects. Thus Microwave Control is a composite component with groups the two tasks, Microwave Oven Control and Oven Timer, which access Oven Data. Microwave

Display Interface is a composite component that groups Display Interface with Display Prompts.

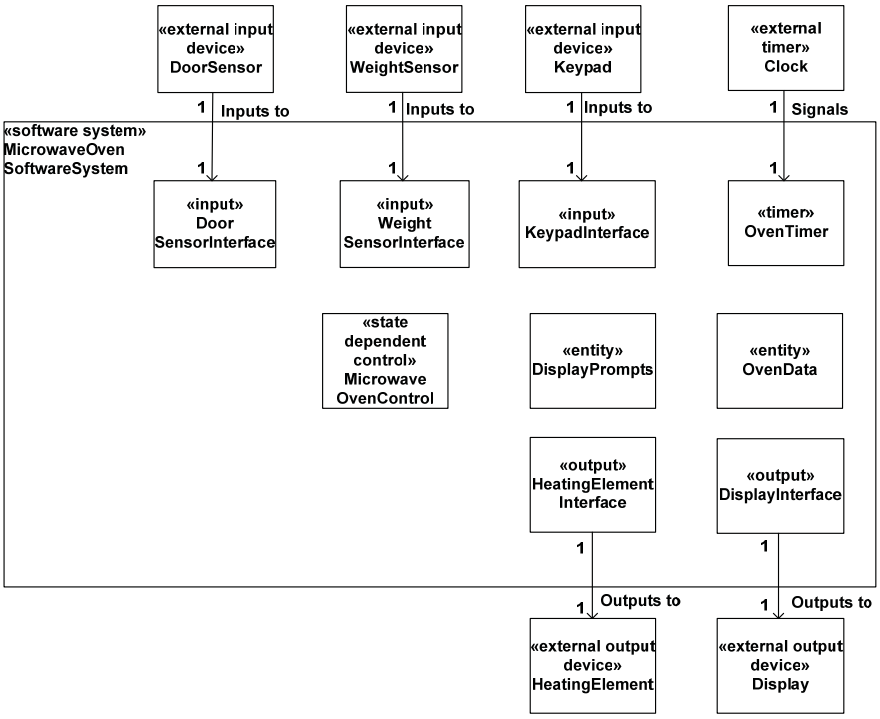

Fig. 8 Software components for Microwave Oven Software System

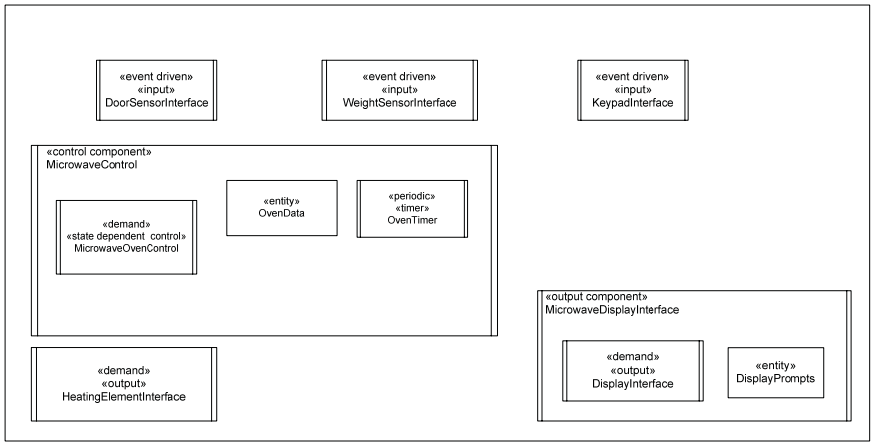

Fig. 9 Concurrent components for Microwave Oven Software System

5.4 Modeling and Design of Input/output Tasks

This section describes the concurrent modeling and design of the input/output tasks, since these tasks directly interface to and communicate with the hardware devices.

An event driven I/O task is needed when there is an event driven I/O device to which the system has to interface. The event driven I/O task is activated by an interrupt from the device, performs the I/O operation and then waits for the next interrupt. An example is given on the UML communication diagram in Fig. 10 in which the Door Sensor Interface event driven input task is awakened by an interrupt from the Door Sensor event driven external input device. This diagram uses the UML notation for active objects for the Door Sensor Interface task and the Microwave Control demand driven task.

Fig. 10 Event driven input task and demand driven control task for Microwave Oven Software System

In the case of a passive device that does not generate interrupts, a periodic I/O task is developed to poll the device on a regular basis. The periodic I/O task is activated by a timer event, performs an I/O operation, and then waits for the next timer event. An example is given on the UML communication diagram in Fig. 11 in which the Temperature Sensor Interface periodic input task is awakened by a timer event from the Digital Clock, and polls the Temperature Sensor passive external input device.

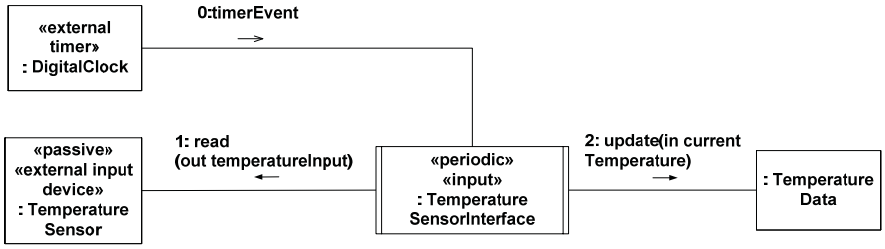

Fig. 11 Periodic input task for Microwave Oven Software System

6. Conclusions

This paper has described an approach for the integration of system modeling and software modeling, This approach is particularly useful for embedded systems, which are software intensive systems that consist of both hardware and software components. This paper has described a modeling solution to this problem with an approach that integrates system modeling using SysML with software modeling using UML for the development of embedded systems using both structural and behavioral modeling. In particular this paper has concentrated on the hardware/software boundary of a system with the decomposition into hardware and software components, and designing the interface between hardware and software components. The modeling approach described in this paper can also be extended to address the performance requirements of embedded systems [11] and to model system and software product lines [12].

7. References

1. Buede, D.M. *The Engineering Design of Systems: Methods and models*. New York: Wiley (2000)
2. Sage, A. P. and Armstrong, J. E., Jr., *An Introduction to Systems Engineering*, John Wiley & Sons (2000)
3. Booch**,** G. et al. *Object-Oriented Analysis and Design with Applications*, 3rd ed. Boston: Addison-Wesley (2007)
4. H. Gomaa, "Software Modeling and Design: UML, Use Cases, Patterns & Software Architectures", New York: Cambridge University Press (2011)
5. M**.** Blaha and J. Rumbaugh**,** *Object-Oriented Modeling and Design with UML*. Upper Saddle River, NJ: Prentice Hall (2005)
6. Douglass, B. P. *Real Time UML: Advances in the UML for Real-Time Systems*, 3rd ed. Boston: Addison-Wesley (2004)
7. S. Friedenthal, A. Moore, and R. Steiner, A Practical Guide to SysML: The Systems Modeling Language, Morgan Kaufmann (2009)
8. Booch**,** G**.,** J**.** Rumbaugh**,** and I**.** Jacobson. *The Unified Modeling Language User Guide*, 2nd ed. Boston: Addison-Wesley (2005)
9. Rumbaugh**,** J**.,** G**.** Booch**,** and I**.** Jacobson**.** *The Unified Modeling Language Reference Manual*, 2nd ed. Boston: Addison-Wesley (2005)
10. Harel, D. and E. Gary, "Executable Object Modeling with Statecharts", Proc. 18th International Conference on Software Engineering, Berlin (1996)
11. H. Gomaa, "Designing Concurrent, Distributed, and Real-Time Applications with UML", Boston: Addison Wesley, (2000)
12. Gomaa, H., Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures. Boston: Addison-Wesley (2005)