Deriving Object Models in Embedded Systems:
A Hierachical Modular Component-Based Approach

Mohamed T. Kimour, Djamel Meslati
Department of Computer Science,
LRI, University of Annaba
Bp. 12, Annaba, Algeria
Algeria
Tel / Fax: 213-38-87-27-56

Abstract: Embedded systems are predominantly control dominated systems and usually designers specify them using state-oriented models, such as FSMs or Petri Nets [13]. However, for modeling more aspects of the systems namely, data and function, it is critical to consider the use of multiple-view models. Especially, identifying and modelling objects are a key activity which is hard and critical, since there is no one-to-one mapping between use cases and objects. In this paper, we provide a new approach that firstly build an hierarchical modular components for the system, and secondly based on simple procedure we how to derive and model the collaborating objects. We argue that our approach enables enriching the use case model and producing more complete requirements.

Key-Words: Use cases, Requirements specification, Statechart, Object model.

1 Introduction

Embedded systems are predominantly control dominated systems and usually designers specify them using state-oriented models, such as FSMs or Petri Nets [1,3,4,7,13]. However, for modeling more aspects of the systems namely, data and function), it is critical to consider the use of multiple-view models. For this purpose UML [1] was adopted, since it is a notation that covers the most relevant aspects of systems and is an industrial standard.

UML is a general purpose modeling language for specifying, visualizing, constructing and documenting the artefacts of computer-based systems, as well as for business modeling and other non-software systems [3,5,6,8,9].

UML presents many advantages for modeling embedded systems at the system-level [17,22,21,16,15,4] . It is a standard, can be used communicate with the customer, is suitable to object oriented design, is totally platform independent, and possesses an extension mechanism to deal with non-standard modeling issues that is being used to define various UML application-domain profiles.

Among its disadvantages, UML can be criticized for having too many diagrams, to not have a precise semantics, and for introducing a new layer in the project [11,12,15,14,18,19,20].

In this paper we provide an approach to identify objects using the architecture component approach, by providing a more encompassing perspective in which design of engineering systems takes place. The levels of system specification provide a way to understand what design is about, namely going from behavior to structure. We found that use-cases can be viewed as a means to identify coherent subsets of the I/O behavior that we desire for the to-be-constructed system.

An architecture for embedded systems suggested four types of objects that need to be in most engineering designs (state, interface, coordination and control objects) and that these objects carry out a process by which sensory information from the environment is processed and analyzed in an upward direction and becomes the basis for decisions that flow downward by successive decomposition of higher level goals into actuatable commands.

We saw that knitting these objects together via collaboration diagrams was, in effect, providing the coupled system specification that we need in order to specify the structure of the system.

This representation facilitates a hierarchical decomposition of the system during analysis, and based on this decomposition, a hierarchical, stage-wise construction process, during design and implementation [13]. We discuss a graphical representation technique that supports such hierarchical decomposition. Then we show how to combine this architectural approach with UML concepts as an alternative approach to integrating systems methodology and object orientation.

The rest of the paper is organized as follows: section 2 describes our approach to identify objects using the concept of hierarchical modular decomposition of the system, and provide an evaluation of this approach with

respect to some related works. Finally, we conclude and present the future work at section 3.

2 Systems Hierarchical Modular Composition Framework

The framework deals with components, or parts, of system. Such components are to be implemented as modules. Modules evidence modularity, by which we mean that modules are self contained and can stand alone or be incorporated, as components into a larger system. There are two types of components: atomic modules and coupled modules.

- Atomic modules are taken off the shelf or are developed in code they are the ground level elements from which all systems built. They have input and output ports.
- Coupled modules are constructed from atomic modules by coupling them together using their input and output ports.

Coupled modules have the same input and output port interfaces as atomic modules and can be treated in the same manner as far as their external relations to other components. In particular, coupled modules can become components in larger systems, just as atomic by adding in a coupling specification to a set of modules, we get a coupled module. By using this module as a component in a larger system with our components, and adding coupling information, we get a hierarchical coupled module. The hierarchical modular approach to the elevator control system defines the atomic modules, DoorControl, MotorControl, elevator and coordinator. Input ports, such as floorStatusIn, designate particular locations where inputs can arrive and be processed by a module. Output ports are locations from which a module can send messages. Ports can be typed as well, so that e.g., floorStatusIn, can handle data in the form of an integer indicating the last floor at which the elevator stopped.

All interaction with the outside world or other modules is required to occur through the modules input and output ports. This requirement allows a module to be truly modular in that it can stand alone as an independent object. On the other hand, a module can be incorporated into a larger system by coupling its ports with those of other components. Such coupling involves connecting output ports to input ports,. e.g., the elevStatusOut output port of the elevator, is coupled to the input port elevStatusIn, showing that elevator status information is passed on to the coordinator through the latters input port.

The hierarchical modular decomposition/construction approach is applicable in principle to design any system. When applied to software it gains computational advantages in combination with object oriented technology. The combination comes about by adopting the hierarchical approach to derive the initial breakdown of the system and then assigning objects to realize the required behaviors of the components.

To see how this works, let's recall the four types of objects that form the basis of engineering systems design. As illustrated in Figure 4, the primary information flow among these objects is as follows:

- state information objects record raw data derived from sensors
- result objects process and store more refined information derived from the state objects needed for planning and control
- planning objects further process the information derived from result objects and formulate plans for system operation to meet given goals
- control model objects carry out the plans by issuing the appropriate commands to actuators.

In initial analysis it is helpful to develop a table (table 1) in which objects are categorized according in the preceding manner along with their particular responsibility in the overall design. Table 1 illustrates this idea for the elevator example. Based on the hierarchical decomposition and using table we derive the object model depicted in figure 2. This was done by applying a separation of concerns principle [10].

3 Conclusion

In this paper we have presented a systematic approach for transition from the use cases to the object model in the embedded systems area. Our approach presents a technique for converting use cases into statecharts and uses the latter as a means to identify objects. The semi-formal nature of statecharts allows for discovering the necessary objects and their properties (operations and attributes), which are needed for realizing the use case. The derived statecharts allow the developers for uncovering ambiguities, omissions, impreciseness, and inconsistency that may be present in the natural language description of the use case.

In this way, while preserving the advantages of the use cases' natural language description (expressivity and ease to use), we also allow for using existing tools to verify and prove some properties of an embedded system.

Currently, besides the development of a supporting tool for our approach, we are investigating the subject of modifying the XMI DTD to represent our extended statechart by means of XML documents, in order to automate the transition between the use case model and the interaction diagrams via statecharts.

Table 1 Object categorization and responsibility table

object	**Category** {sensor = se actuator = ac state info object = sio result object = ro planning object = po control model object = cmo }	**Responsibility**
Door sensor	se	sense position of door
Floor sensor	se	sense location of elevator
FloorButtons	se	provide open Calls
CallButtons	se	provide DestinationFloors
DoorStatus info	sio	record state: open/closed or blocked
ElevatorFloor	sio	record current floor
ElevatorStatusInfo	ro	compute last floor, stopped or moving direction (up/down) from ElevatorFloor, DoorStatus and motor
OpenCalls	po	provide destinations called from outside
DestinationFloors	po	provide destinations called from inside
DoorControl	cmo	send commands to door
MotorControl	cmo	send commands to motor
Coordinator	cmo	decides on direction and next floor/when to depart
LightSwitches	ac	button lights provide information to passengers
Motor	ac	drives the elevator
Door	ac	opens/closes door

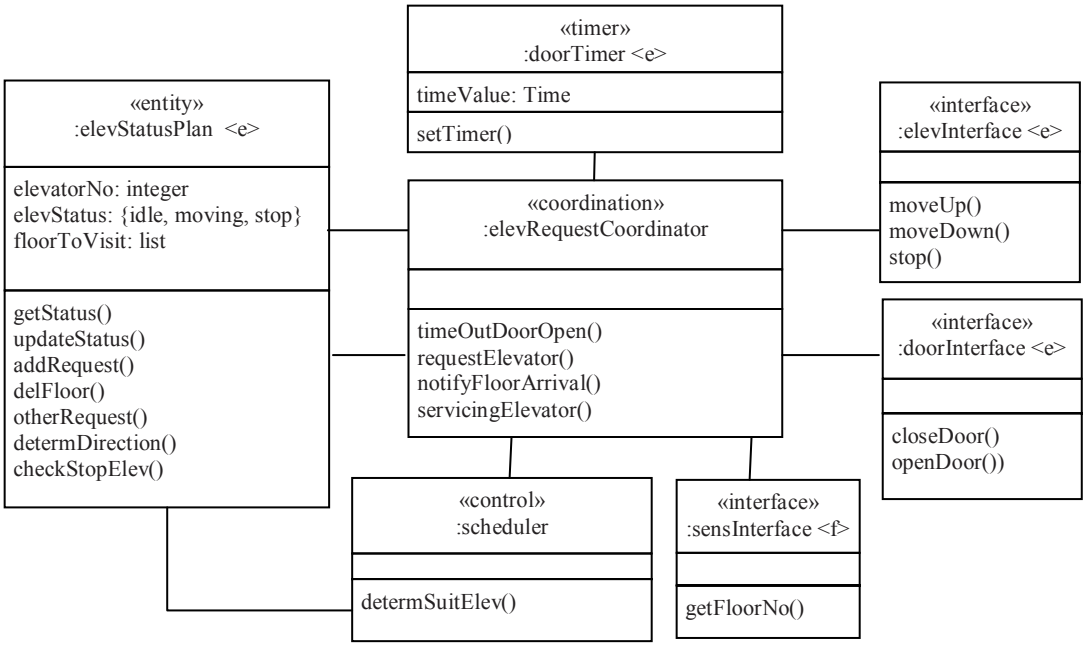

Fig.1 The derived object model.

References:

[1] G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User's Guide*, Addison Wesley, 1999.

[2] A. Cockburn, Structuring Use Cases with Goals, *Journal of Object-Oriented Programming*, September-October 1997 (part I) and November-December 1997 (part II).

[3] B. P. Douglass, *Real-Time UML: Developing Efficient Objects for Embedded Systems*, Addison-Wesley, 2nd Edition, 2000.

[4] J. M. Fernandes, R. J. Machado, From Use Cases to Objects: An Industrial Information Systems Case Study Analysis, *OOIS'01 (7th International Conference on Object-Oriented Information Systems)*, Springer-Verlag, 2001, pp. 319–328.

[5] M. Glinz, S. Berner, S. Joos, J. Ryser, The ADORA Approach to Object-Oriented Modeling of Software, *CAISE'01 (13th Conference on Advanced Information Systems Engineering) LNCS*, Interlaken, Vol. 2068, June 2001, pp.76-92.

[6] H. Gomaa, *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Addison-Wesley, 2000.

[7] D. Harel, Statecharts: A Visual Formalism for Complex Systems, *Science of Computer Programming*, Vol. 8, 1987, pp. 231-274.

[8] I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[9] I. Jacobson, M. Christerson, P. Jonsson G., Övergaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.

[10] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V.Lopes, J.-M. Loingtier, J. Irwin, Aspect-Oriented Programming, *ECOOP'97 (11th European Conference Object-Oriented Programming), LNCS*, Vol. 1241, 1997, pp. 140–149.

[11] Y. Liang, From Use Cases to Classes: a Way of Building Object Model with UML, *International Journal of Information Software and Technology*, Vol. 45, No 2, 2003, pp. 163-180.

[12] D. Liu, K. Subramaniam, B.H. Far, A.Eberlein, Automatic Transition from Use Cases to Class Model, *IEEE/CCGEI'03*, Montréal, Canada, May 2003.

[13] L. Maciaszeck, *Requirements Analysis and System Design*, Addison-Wesley, 2001.

[14] D. Rosenberg, K. Scott, *Use Case Driven Object Modeling with UML: A Practical Approach*, Addison-Wesley, 1999.

[15] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Larenson, *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[16] J. Ryser, M. Glinz, A Scenario-Based Approach to Validating and Testing Software Systems Using Statecharts, *ICSSEA'99 (12th International Conference on Software and Systems*

Engineering and their Applications), CNAM, Paris, France, 1999.

[17] B. Selic, Using UML for Modeling Complex Real-Time Systems, *LNCS*, No 1474, 1998, pp. 250-262.

[18] S.S. Somé, Beyond Scenarios: Generating State Models from Use Cases, *ICSE'2002 (International Workshop on Scenarios and State Machines: Models, Algorithms and Tools)*, Orlando, May 2002.

[19] I. Sommerville, *Software Engineering*, 6th Edition, Addison-Wesley, 2001.

[20] R.S. Wahano, B.H. Far, A Framework for Object Identification and Refinement Process in Object-Oriented Analysis and Design, *ICCI'02 (1st International Conference on Cognitive Informatics)*, Calgay, Canada, 2002.

[21] M. Sveda, R. Vrba, Embedded System Specifications Reuse by a Case-Based Reasoning Approach, *WSEAS Transactions on Computers,* Vol.2, No.1, January 2003.

[22] M. Goudarzi, Dh. Hessabi, Synthesis of Object-Oriented Descriptions Modeled at Functional-Level, *WSEAS Transactions on Computers,* Vol.2, No.1, January 2003.