Synchronized Execution Sequence Based Software Architecture for Object-Oriented Embedded Systems

Toshiaki Aoki[†], Akira Kawaguchi[††], Tomoji Kishi[††], Takuya Katayama[†]

[†] Japan Advanced Institute of Science and Technology
[††] NEC Corporation

Abstract

According to the increase of size and complexity of embedded software, object-oriented software methodologies are going to be adopted in this domain. The OO methodologies so far developed, however, mainly focus on analysis phase of software development and do not pay much attention on design phases where the embedded software is different from the usual information systems. In the embedded systems, hardware restrictions and time constraints are very severe and we need to take these factors into consideration in the earlier stage of the design phase where logical structures extracted in the analysis phase are transformed into software structure.

In this paper we propose a software architecture for building embedded systems using object-oriented methodologies. This architecture is based on the concept of synchronized sequence of method execution which is derived from object interaction diagrams found in most OO-methodologies. This architecture facilitates to estimate timing requirements of the constructed systems and was successfully adopted in an industrial setting in prototyping a real product.

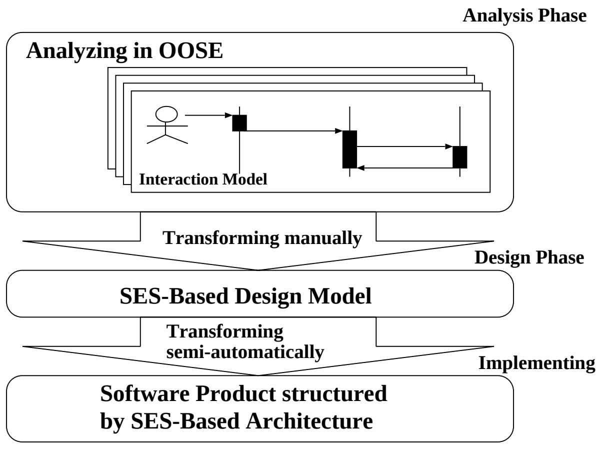

Figure 1: Overview of SES Approach

1 Introduction

Embedded software put into industrial and consumer products is becoming bigger and bigger with the advances of hardware technologies and increase of services they have to offer. This makes their developer to switch their development methods to object-oriented methods which have been successfully used for large scale information systems.

However, development of embedded systems requires their specific properties to be taken into consideration such as time and hardware constraints. This makes the straightforward adoption of object-oriented methodologies difficult as these constraints have to be considered in design phase rather than implementation phase. To solve this problem we need a design model which is suited for the embedded system domain and is able to represent such non-functional properties, and this model has to be converted into software architecture which can be effectively realized on available software/hardware infrastructures.

In this paper we propose a software architecture for embedded software which is called a *synchronized execution sequence based architecture*, abbreviated as *SES-based architecture*. This architecture represents software as a collection of sequences of method execution which is not blocked due to waiting hardware and resource availability. This architecture will make analysis of timing requirements easier than other architectures based on the direct behavior of objects. Also, this architecture could be directly constructed from object interaction diagrams obtained in analysis phase in OO methodologies such as OOSE[2].

The concept of SES has been successfully adopted in developing a telephone system at NEC corporation[3]. This paper tries to formalize SES and propose a software architecture base on SES. In Chapter 2, we introduce SES concept using a small example in the telephone system. Chapter 3 introduces SES model and explain how it is constructed from interaction model developed in the analysis phase. Chapter 4 defines SES-based software architecture assuming the usual RTOS implementation infrastructure.

2 Synchronized Execution Sequence

We introduce here SES concept using an example which appeared in the development of a telephone system at NEC corporation. The system has been developed adopting OOSE methodology, where USECASE, or object interaction diagram plays a central role. Of course, these models represent logical structures of the target system and do not involve timing or resource information.

Consider an interaction diagram in Figure2, where PLL and RF are object performing phase lock looping and radio wave transmission respectively. Though these objects represent logical entities in the system, we need to consider, in the design phase, that the time PLL object returns *Finished PLL set* event after receiving *Requiring PLL set* event is very long compared to other operations, as it has to wait for PLL hardware to finish its phase locking operation. So, if we divide the operation performed by PLL object into two operations which are executed before and after the wait operation respectively, we can organize the entire operations into two execution sequences which will not be blocked by waiting PLL hardware to finish its operation. These sequences are called *Synchronized Execution Sequences* and are executed efficiently in the usual computational environment with the help of an *coordinator* which manages correct invocation of SES.

A prototype telephone system has been developed using SES concept. Realtime constrains were easy to satisfy as SES is never blocked and its timing properties are analyzed without much difficulty. Though memory requirements were not met when every SES is mapped to a task in a realtime OS, this was also solved by merging tasks. The final system required 10 percent additional processing time and memory compared to the one developed by a conventional method.

3 SES-based Design Model

Logical structures constructed in the analysis phase have to be transformed into design models considering

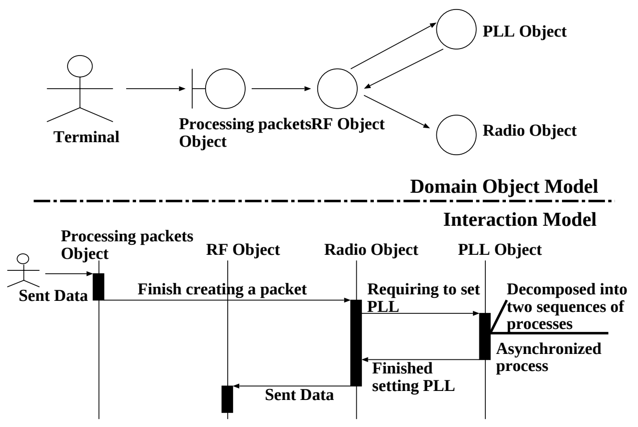

Figure 2: Analysis Model of a Telephone System

hardware and resource constraints. Especially, in embedded system development, these constraints are more strict than the usual information systems and the design models have to allow to incorporate such constraints directly. To this end, they should be based on operational characteristics of available computer hardwares and operating systems. SES is an abstraction of threads which will run unblocked under the control of some central coordinator which plans their invocation. In this paper we assume that OOSE like methodology will produces object interaction models as the result of analysis phase.

3.1 SES

SES is defined as a sequence of event communications and related processes in objects subject to the following conditions.

Condition 1: Any process in a SES is executed sequentially.

Condition 2: Any process in a SES is executed synchronize with the succeeding operation in the SES.

An element of SES is represented as $e_1.o_1 \rightarrow o_2.p_2$ and denotes a fact that upon receiving an event e_1 form object o_1, object o_2 executes a process p_2. SES is then represented as a sequence of such elements:

$$e_1.o_1 \rightarrow o_2.p_2 \; e_2.o_2 \rightarrow o_3.p_3 \; \cdots \; e_{n-1}.o_{n-1} \rightarrow o_n.p_n$$

SESs are derived from interaction diagrams. Consider the interaction diagram in Figure3 and try to extract SESs from it. We assume the processes $p2$ and $p3$ could be executed in parallel and execution of $p3$ involves hardware operation which take a long time. In this case,

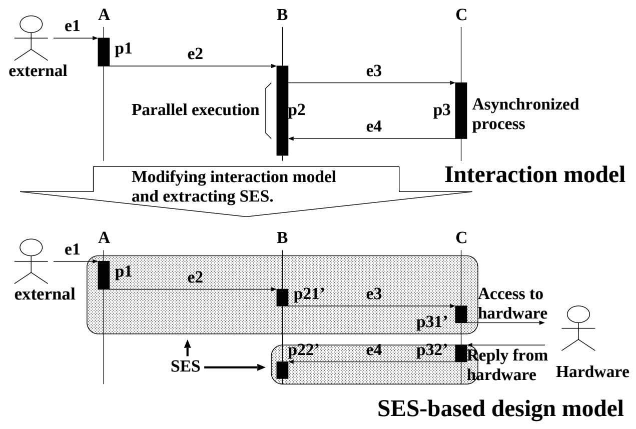

Figure 3: Extracting SES from Interaction Model

$p2$ and $p3$ are split into $(p21', p22')$ and $(p31', p32')$ respectively and the following two SESs are obtained.

$$e1.external \rightarrow A.p1\ e2.A \rightarrow B.p21'\ e3.B \rightarrow C.p31'$$
$$reply.hardware \rightarrow C.p32'\ e4.C \rightarrow B.p22'$$

SESs derived are not unique and the following SESs are also obtained from the same interaction diagram.

$$e1.external \rightarrow A.p1\ e2.A \rightarrow B.p2$$
$$e3.B \rightarrow C.p31'$$
$$reply.hardware \rightarrow C.p32'$$

Choice among possible SES structures is, of course, an important design decision and will be made on design/implementation constraints and performance considerations.

3.2 SES-based Design Models

A SES specifies a fraction of computation in the system under consideration. Usually the system involves multiple SESs which are invoked repeatedly. SES-based design model is introduced to represent possible computation in the system and defined as a set of SESs whose invocation in controlled by a *coordinator* whose control structure is described by a state transition diagram. Specifically, with each state of the coordinator is associated a set of SESs which are executed in parallel when the coordinator is in the state. State transition occurs when an event is created from some of the SESs. In Figure4, three SESs $ses1, ses2, ses3$ are executed in parallel in the state $s1$. If an event $e1$ is created from one of them, the coordinator makes state strantition from $s1$ to $s2$ where two SESs $ses4$ and $ses5$ are allowed to execute.

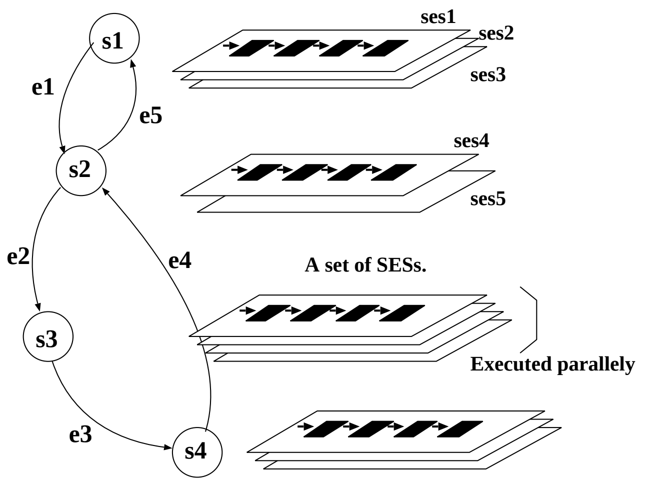

Figure 4: SES-based Design Model

A *SES-based design model* UM is defined as follows.

$$UM \stackrel{def}{=} (SES, ST_{sys})$$

where SES is the set of SESs under consideration and ST_{sys} is a state transition diagram representing the coordinator which controls execution of SESs.

$$ST_{sys} \stackrel{def}{=} (S_{sys}, E_{sys}, T_{sys}, s_{sys}, \mathcal{M}_{sys})$$
where E is a set of events sent from SES,
S_{sys} : a set of states,
$T_{sys} : S_{sys} \times E_{sys} \rightarrow S_{sys}$,
$s_{sys} \in S_{sys}$(an initial state),
$\mathcal{M}_{sys} : S_{sys} \rightarrow Pow(SES)$

The mapping $\mathcal{M}_{sys}$ assigns to a state s SESs which are executed in the state s. They are executed in parallel.

3.3 Access Control over Resources

In general, SESs attached to a state run in parallel and they may access to a resource simultaneously and we need access control mechanisms such as semaphore to avoid it. Introduction of this kind of mechanism will, however, cause a blocked status of SES execution and makes its timing estimate difficult. This also consumes precious time resource of the system.

In the practical embedded system design, this problem is solved by carefully scheduling and planning the use of the resource and try to not use such access control features dynamically. We followed this strategy and introduced the coordinator to control the execution of SESs so that their execution is not blocked by waiting to access to resources or undesirable racing of processes will

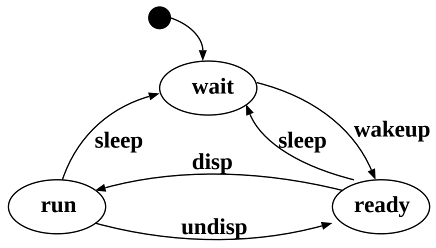

Figure 5: State transition diagram of task

not occur. Though this will decrease the description power of our model, it will increase its analyzability which is considered more important in designing reliable embedded systems.

4 SES-based Software Architecture

SES design model is realized by a software architecture that consists of SES components and coordinator component. We describe how these components are built assuming a popular implementation environment.

4.1 Implementation Environment

Here we will briefly describe an implementation environment to run SES-based software. We assume a realtime operating system (RTOS) is available in the environment, where tasks are units of execution and their invocation is controlled by a scheduler. Though there are a variety of RTOSs and their features, we assume only a small fraction of them which will be common to all of them.

4.1.1 Tasks

We assume tasks have three states *running*, *wait* and *ready* and they have conventional meaning. The states are switched by receiving events (Figure5).

The events *sleep* and *wakeup* are issued by tasks, and *disp* and *undisp* are sent from scheduler to activate/suspend tasks.

4.1.2 Scheduler

Scheduler sends *undisp* event to suspend running task and *disp* to activate a ready task. It will use a scheduling algorithm to select a ready task and also it will have mechanisms to control priority of tasks. We are not interested in these things here as our architecture do not assume specific scheduling algorithms and mechanisms.

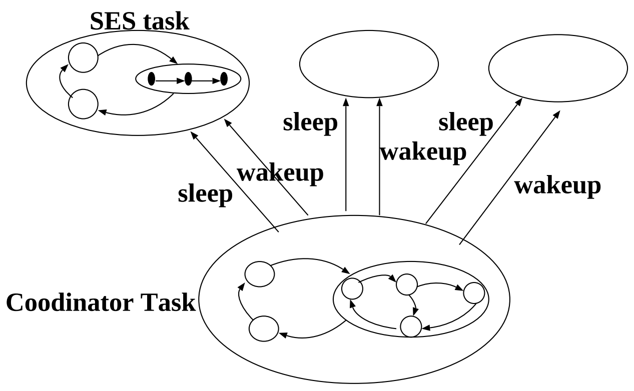

Figure 6: Overview of SES-based Architecture

4.2 SES-based Software Architecture

SES-based design models are implemented by SES-based software architecture consisting of SES tasks and coordinator task which run on a RTOS(Figure 6).

The coordinator task is an implementation of the state diagram in the design model. It sends the events *sleep*, *wakeup* to SES tasks to control their activation according to the state diagram.

4.2.1 Architecture of SES

SESs in the design model are implemented as SES tasks in the SES-based software architecture.

Processes in SES are implemented as methods of objects and the role of the SES is realized by a sequence of these method calls. The task needs an additional control for the sequence of the method calls, which we call *implicit control object* as this part does not appear explicitly in the interaction diagrams obtained in the analysis phase(Figure 7). In the implicit control object, the following three things are done.

1. Periodical monitoring of events sent to SES
2. Doing processes in SES
3. Notifying results to the coordinator task

The implicit control looks like:

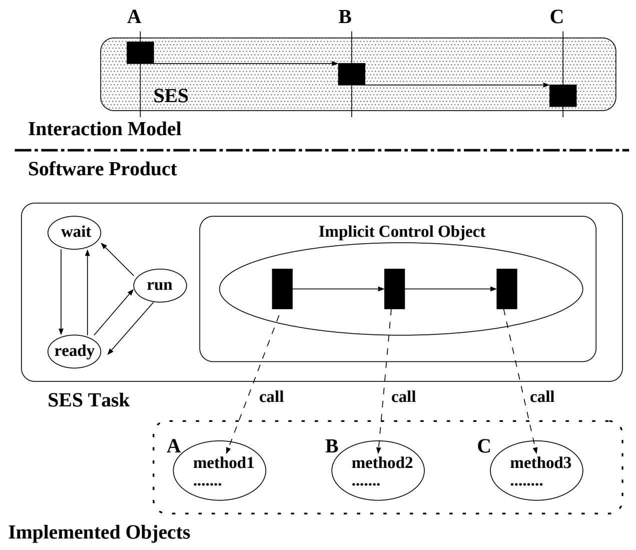

Figure 7: Aechitecture of SES Task

```
implicit_control_object {
 while(1){
   while(watch ()); /* if actor is not active,
this task is sleep for a moment. */
   B.method1() ;
   C.method2() ;
   D.method3() ;
   sendMessage(coordinator_task, message);
  }
}
```

Usually, we need to add additional control structures to the above code as we only extract typical and partial structure of the system in forming interaction diagrams and exception handling features are not usually captured in the analysis phase.

4.2.2 Architecture of Coordinators

The coordinator task is just an implementation of state diagram of the SES-based design model. It waits for a completion event from SESs of the current state and send out *sleep* event to them and *wakeup* event to SESs of the next state.

5 Conclusion

A software architecture for implementing object-oriented embedded systems is introduced in this paper which is based on the concept of synchronized sequence of method execution. SES-based architecture allow us to (1) derive programs naturally from interaction diagrams in object-oriented analysis method like OOSE, (2) decrease wasteful time spent in RTOS such as for busy waiting, and (3) facilitate to estimate execution time. These properties are very important in building embedded systems which are usually time critical.

This architecture is emerged from an experience of developing a telephone system. One of the key issues in implementing embedded system is to build the system within a give hardware cost. In the current method, one task is generated from one SES and this usually increases the number of tasks and increases the size of memory required. In the telephone system development, multiple tasks are manually merged into a single task to avoid this problem. We need to study to optimize the number of SESs given a set of interaction diagrams.

We consider SES-based implementation has advantage in estimating execution time, however, current model does not have features to represent timing information explicitly . We need to augment our current model with timing information.

References

[1] Shaw,M., Garlan, D.: Software Architecture, Prentice Hall, 1996.

[2] Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.: Object-Oriented Software Engineering, Addison-Wesley, 1992.

[3] Kawaguchi, A., Kishi, T.: Object-Oriented Design Methodologies for Embedded Systems, NEC Technical Journal, Vol.50, No.12, pp.28-34, 1997.