Framework-based Embedded Real -Time System Development

Hui-Ming Su., Jing Chen
E-Mail: ken@rtpc06.ee.ncku.edu.tw
jchen@mail.ncku.edu.tw
Department of Electrical Engineering,
National Cheng Kung University, Taiwan R.O.C.

Abstract

The development of an embedded real-time system is a complicated task. Not only the users' requirements are difficult to capture but there are also the stringent timing constraints on critical tasks. Reusing the successful developing experience is a shortcut. In this paper, a framework-based approach is presented to help efficient design embedded real-time systems. The approach is based on an object-oriented framework, which is a collection of collaborating components that provide a set of services for a given domain. Our framework construct consists of three packages, namely UsrApp, RTService and OSAbstract, to aid designers customize their applications derived from the abstract classes in these packages.

Keywords: framework, UML, package, active class, reactive class

1. Introduction

Object-oriented (OO) technology employs encapsulation and inheritance as basic reuse techniques. In a system, objects are identified, encapsulated, and positioned in a hierarchy of classes by taking into account their dependency. Class libraries are thus the basic structures for reusing objects. System designers have found such libraries to be limited in their reuse capability due to their generality. Recently, several application-domain techniques have been proposed, which significantly improve the degree of reuse. Ordered in the ascending order of their degrees of reuse, design patterns, components, and object-oriented application frameworks are widely used reuse techniques.

Design pattern is a problem-solution pair that captures successful development strategy. Patterns aid in reusing successful design strategies by giving a more abstract view to concrete design strategies. A component acts as a black box allowing designer to reuse it through knowledge of its interface only. Components are self-contained instances of abstract data types that can be integrated into complete applications.

An object-oriented application framework (OOAF) is a reusable, “semi-complete” application that can be specialized to produce custom applications [11]. Within the OO arena, OOAF are application-domain specific reuse methods, which have been proposed for general-purpose systems. However there are relatively little works on application framework for the design of an embedded real-time system. Motivated by this, we propose a framework-oriented concept to build an embedded real-time system. This framework construct consists of a group of components: Analyzer, Task Composer, Scheduler, OS Manager and Code Generator. It helps the developing process of embedded real-time system from requirement analysis through code generation.

In the following, section 2 discusses related works. Section 3 explains the concept and overview of this framework construct. Section 4 describes the implementation of a tool based on the concept of this framework. This tool supplies a friendly user interface for the designer to model an application system. Finally, section 5 summaries this work and briefly discusses future works.

2. Related Works

There have little works on embedded real-time system development using framework. Object-Oriented Real-Time System Framework (OORTSF) [12] is a simple framework like class library only lists the classes used in real-time application development. It is difficult comprehension of the collaboration among the classes and developing an application using it.

RTFrame [9] is an application framework solution developed especially for real-time system design. It consists of five components: Specifier, Extractor, Scheduler, Allocator, and Generator. Within RTFrame, several design patterns have been proposed for real-time systems. It has a clear process for designing an embedded real-time system. But the relationship between abstract classes is not clear. RTFrame is not easy to extend.

Rhapsody [1] is a commercial development tool for embedded real-time system development with a framework inside. It has defined a completely framework for code generation of embedded real-time system, but lacks of schedule class and error handler.

We therefore suggest a new framework with schedule class and error handler class for embedded real-time development. Using this framework shows a significant increase in design productivity through design reuse and reduction in both design time and effort.

3. Framework Construct

The concept of framework is expressed in figure 1 using UML notations. It is a composite package consists of three logic packages, which are UsrApp, RTService and OSAbstract. The UsrApp is the base class collection for application domain specific classes to inherit. Time and resources management are the critical issues in real-time system. The RTService has well defined structures for real time specific classes to inherit. The OSAbstract wraps the operating system service to aid application domain classes and real-time specific classes.

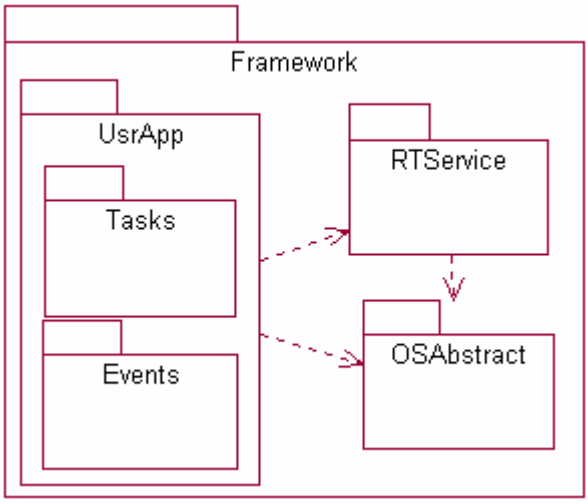

Fig. 1: The Architecture of the Framework

3.1 UsrApp

The UsrApp package consists of a set of collaborative classes that form the fundamental architecture of a reactive, multithreaded system. It is designed for implementing subclasses to instantiate the domain specific application objects. An active object is an object that owns a thread and can initiate control activity, which runs on its own task (thread), with a message queue available on the object. A reactive object is one that has a mechanism for consuming events and triggered operations. It has a clear lifetime whose current behavior is affected by its past [5]. The application domain specific objects are made of active objects, reactive objects and main program showing in figure 2.

3.1.1 Tasks

The tasks package consist three abstract classes, which is FWThread, FWReactive and FWMain. A thread is represented by FWOSThread, which wraps an operating system thread belonging to the OSAbstract package. FWThread contains code that manages an event queue. It executes an infinite

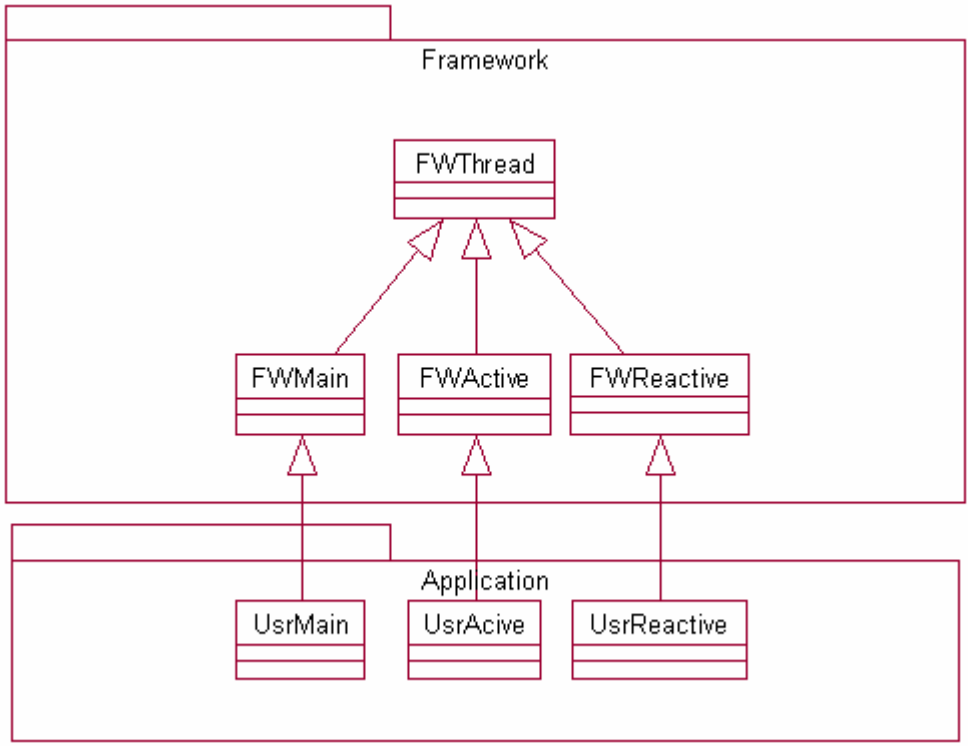

Fig. 2: The Architecture of Application Domain Objects

event dispatching loop, taking events from the queue and injecting them to the target objects. The FWThread class is the base class in the framework for each active class. User active classes inherit from FWThread runs an event loop on its own thread and dispatches events to client reactive classes.

RWReactive is the framework base class for all reactive objects. Reactive objects can process events via statecharts transitions. Essentially, a reactive class is one that reacts to events; that is, it is an event consumer. Each reactive class must associate with an active class, from which its events are dispatched.

The FWMain class is a special case of FWThread. It defines the default active class for an application. The FWMain inherits from FWThread and only one instance is created. It represents the main program.

3.1.2 Events

There are three types of events, signal events, triggered operations and timeout events. The FWEvent class is the base class for all events. Events can trigger transitions between states in statecharts and activity diagrams.

FWTimeouts are a specialization of class FWEvent. The timeout class derives from FWTimeout implements timeouts issued by statecharts or activity diagrams within reactive classes. The system timer manages the timeouts and sends them to the reactive object. The timeouts objects are created either entering states with timeout transitions or delay requests from user code.

The FWTimerManager is responsible for managing the timeout. It is an additional thread that provides timer support for the application. The FWTimerManager class manages timeout requests and issues timeout events to the application objects. FWTimerManager is a singleton object in the execution framework. The FWTimerManager has a timer, which notifies it periodically whenever a fixed time interval has passed. At any given moment, the FWTimerManager holds a collection of timeouts that should be posted when their time comes.

A reactive object has function members. A sender object to send an event to a receiver object uses the genEvent. The takeEvent is used by the event loop (within the thread) to make the reactive object process an event. The comsumeEvent is the main event consumption method. The FWThread function member, exEvent is the thread main loop function. The QueueEvent queues events to be processed by the thread event loop. For the FWEvent, setDest sets the destination attribute to the reactive object. The getDest determine the reactive object destination for the event.

3.2 RTService

The RTService package consists of a set of collaborative classes that manage the real time requirements. The FWProtected is responsible for the protected resources manager. The FWSchedule is responsible for schedule policy implementation.

A protected object is passive object. The active object must call the service of the FWProtected object. Resources in a class can be monitored, which allows only one operation to access the resource at any given time. A subclass derived from FWProtected can be used to model an exclusive resource: at any given moment, only a single copy of a single guarded operation (of the class) can be executing. The FWProtected class is the base class for all protected objects. It supports the operations lock and unlock using FWOSMutex. One central characteristic of real-time system design is the existence of resources that, in the presence of concurrency, must be managed. The framework includes abstractions for concurrency control mechanisms. FWOSMutex is a wrapper class for an operating system mutex. It supports the operations lock and unlock. A mutex is used for managing exclusive resources.

The FWSchedule is responsible for providing the scheduling code for the scheduled tasks. There are two kinds of priorities: fixed, dynamic. For fixed priority-based scheduling, the priority of a task is decided before running such as rate-monotonic scheduling algorithm [13]. The priority is assigned inverse to its period. For dynamic priority-based scheduling, the priority of a task is decides when running such as deadline first scheduling algorithm [16], which priority is assigned inverse to its remaining time to deadline.

FWErrorHasndle has association relationship with FWSchedule and application objects. It has an attribute, signal, which represents error number. The function member, errHandle processes errors according to signal number. It is a virtual function. The error handler of user can override this function member to extend capability of error handling.

3.3 OSAbstract

The operating system OSAbstract package provides a abstraction layer through which the framework and generated code access operating system services. Each one represents an operating system object. In general, each target environment requires a custom implementation of the OSAbstract.

The AbstractLayer package defines classes that describe basic operations and entities used by the operating system, including FWOSThread, FWOSTimer, FWOSMutex, FWOSEventFlag and FWOSSemaphore. The FWOSThread provides basic threading features. The FWOSTimer acts a building block for OMTimerManager, which provides basic timing services. The FWOSMutex protects critical sections within a thread using binary mutual exclusion. The FWOSEventFlag synchronizes threads. Threads can wait on an event flag by calling wait. When some other thread signals the flag, the waiting threads proceed with their execution. The FWOSSemaphore allows a limited number of threads in one or more processes to access a resource. The semaphore maintains a count of the number of threads currently accessing the resource.

4. A Tool based on the Framework

In order to realize the concept of this framework, a tool is under development. The tool maintains a database of all analysis and design information and offers several views of that database. It supplies a dedicated toolbox for each view to model the target system. The right window is the tree view of all models. The right window is the detail diagrams and notations of the selected model. Figure 3 is an overview of this tool.

The designer models the target system beginning with requirements analysis to capture application domain objects. Use case diagrams are used to express main functions of the target system. Scenarios and statecharts are used to express dynamic state for each use case. Objects and their properties can be identified from use cases and scenarios. The designer representing the object structure of the target system builds the object diagram. The behavior of each object is expressed in the statecharts.

The designer marks active objects and reactive objects in object diagram marking as <<active>> and <<reactive>> stereotype. The task diagram shows only the active objects and reactive objects. Task Composer composes all the tasks recording in the Task-Table. The process of analysis and design is iterative. It means the designer selects the highest risk use case to elaborate first, identifies the including tasks then elaborates the others.

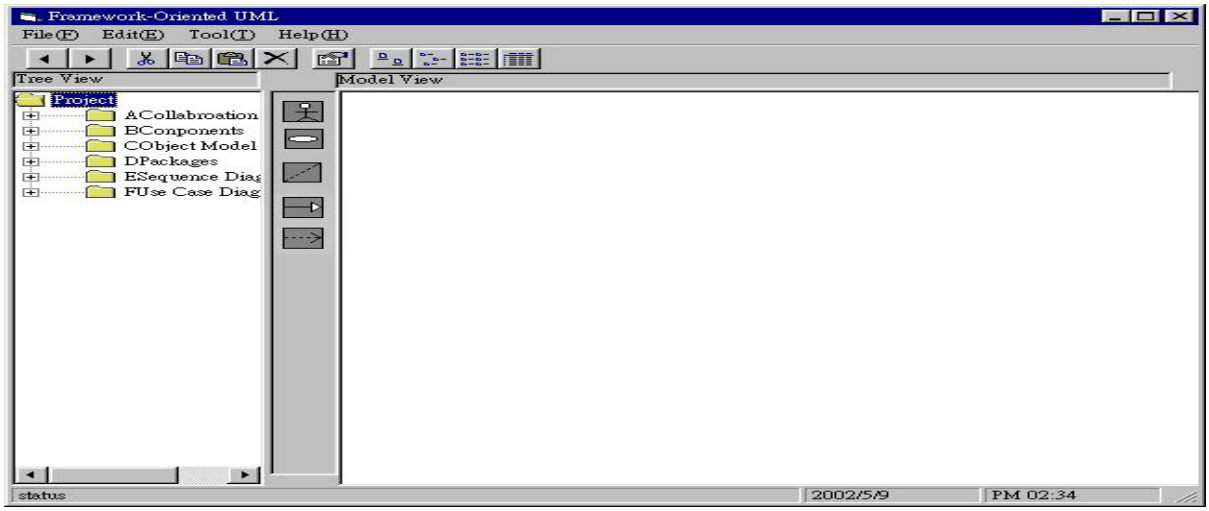

Fig. 3: A GUI of the Framework-Oriented Development Tool

This framework can generate codes merging application objects and operating system services after the scheduling test is successful completely.

5. Summary and Future Works

A framework is a collection of collaborating classes that provide a set of services for a given domain. This paper proposes a framework-based approach to embedded real-time system development. There are three logical packages in a framework. The UsrApp is designed for application domain specific classes to inherit. The RTService is designed for resource management and task schedule classes to inherit. The OSAbstract wraps the operating system service. It helps the designer to save both time and effort during the design phases

A good framework contains classes with well-defined structures and concretely collaboration between these classes. This framework focuses structures and functions description of the abstract classes. In the future, we will enhance the structures of classes and the collaborating relationship between classes to make it more flexible and expansible.

References

[1] "Execution Framework Reference Guide", I-LOGIX home page, http://www.ilogix.com/.

[2] G. Booch, J Rumbaugh, and I. Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley Longman, 1999. ISBN 0-201-57168-4

[3] B. P. Douglas: "Designing real-time systems with UML, parts 1,2 and 3", Embedded Systems Programming, March-May 1998.

[4] B. P. Douglass: "Doing hard-time: developing real-time systems with UML, objects, frameworks, and patterns", Addison-Wesley, 1999, ISBN 0-201-49837-5

[5] B. P. Douglass, "REAL-TIME UML: Developing Efficient Objects For Embedded Systems Secondary Edition ", Addison-Wesley Longman, 1999, ISBN 0-201-65784-8.

[6] M. Fayad and D.C. Schmidt. "Object-oriented application frameworks", Communications of the ACM, Special Issue on Object-Oriented Application Frameworks, 40(10), October 1997.

[7] M. Fowler, "UML Distilled: Applying the Standard Object Modeling Language", Addison-Wesley

Longman, 1997, ISBN 0-201-32563-2.

[8] M. Gergeleit, J. Kaiser, and H. Streich. “Checking timing constraints in distributed object-oriented programs,” ACM OOPS Messenger, 7(1):51–58, January 1996.

[9] P.A. Hsiung, “RTFrame: An Object-Oriented Application Framework for Real-time Application ”, Proceedings of the 1998 IEEE, Technology of Object-Oriented Languages,pp. 138-147, 1998.

[10] I. Jacobson, G. Booch, J. Rumbaugh: “The unified software development process”, Addison-Wesley, 1999, ISBN 0-201-57169-2.

[11]R. Johnson and B. Foote. “Designing reusable classes,” Journal of Object-Oriented Programming, 1(5):22–35, June 1988.

[12] T.-Y. Kuan, W.-B. See, and S.-J. Chen. “An object-oriented real-time framework and development environment,” In Proc. OOPSLA’95 Workshop #18, 1995.

[13] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real time environment. Journal of the Association for Computing Machinery, 20(1):46–61, January 1973.

[14] Michael J. McLaughlin and Alan, “Real-Time Extension to UML” Dr. Dobb’s Journal December 1998.

[15] Terry Quatrani, “Visual Modeling with Rational Rose 2000 and UML,” Addison-Wesley.1999 ISBN 0-201-69961-3

[16] N.C. Audsley, A. Burns , A.J. Wellings, Deadline Monotonic Scheduling Theory and Application, Control Engineering Practice, Vol. 1(1), pp. 71-78 ,1993