

UML层次状态自动机向导 (UML State Machine Wizard)

(东南大学计算机科学与工程系, 南京, 210096)

何伟 金远平

摘要: 随着软件复用技术的发展, 可复用的应用框架日益受到人们的重视。应用框架有助于实现领域内体系结构层次较大粒度的设计复用, 已成为一个软件系统的核心。在对层次状态机基本概念说明的基础上对嵌入式软件开发领域中的层次结构进行了深入研究, 最后设计并实现了一个可复用的应用框架State Machine Wizard, 在嵌入式软件开发中有很强的实用价值。

关键字: 应用框架; 层次状态机; 活动对象; 面向状态; 统一建模语言

1 引言

近年来, 随着软件复用技术的发展, 设计可复用的应用框架变得更加重要。应用框架有助于实现领域内体系结构层次较大粒度的设计复用。目前在嵌入式软件开发中, 通常是针对某个应用问题去开发一个程序。尽管很多程序是相似或相近的, 但每次都要进行许多重复工作。这样的开发模式, 使得开发的软件不仅具有模块化程度低, 可复用性差, 可移植性差, 维护成本高等缺点, 而且具有内在的不完备性和不可靠性。随着应用框架研究的深入, 这种局面正在逐步改变。

层次状态机 (Hierarchy State Machine)^[4]是实现状态建模的一种很好的形式方法。在嵌入式软件开发中, 用层次状态机描述嵌入式软件系统直观、简洁, 而且各状态机之间的层次关系、并发关系十分清晰。因此, 许多嵌入式系统应用程序都被设计成状态机。当 UML(Unified Modeling Language)首先被采用时, 许多嵌入式开发者把 UML 技术合并到静态的 CASE(Computer Aided Software Engineering)技术中去, 用它们来可视化的构建软件体系结构。其中最著名的采用了 UML 方法的 CASE 工具就是 Rational Rose。然而随着技术的不断发展, 像 Rose 这样的 CASE 工具已经不能跟上正在进行的技术变革。模型驱动开发 (Model-Driven Development) 环境提供了清晰的模型以实现代码同步和自动化的仿真, 已经被证明是开发小组跟上新兴技术的关键步骤。基于此种思想, 本文对层次状态机本质和嵌入式软件系统的层次结构进行了深入研究, 然后利用应用框架技术把嵌入式软件开发中与应用领域和操作系统相关的部分抽象出来, 构建一个可复用的适合嵌入式软件开发领域的应用框架。

2 应用框架研究

应用框架是指一个可复用的、部分实现的软件制品, 能够被实例化扩展, 以生成特定的应用。它是一个不完整的系统, 是一个特定领域内的应用程序的部分设计和实现。它能够被裁剪, 从而创建一个完整的应用程序。典型的应用程序框架有: Microsoft Foundation Class(MFC), 它为C++用户提供一套面向对象程序开发的辅助工具, 称之为类向导器(Class Wizard); Delphi 和Visual Basic, 被称之为一种程序快速开发工具, 它们采用属性、方法、事件结构, 是一种基于组件的开发应用框架。但它们都不适合于开发嵌入式系统应用程序。

本文以下部分将介绍如何设计并构建一个适合嵌入式软件开发领域的应用框架State Machine Wizard的关键技术。

2.1 层次状态机

层次状态机 (Hierarchy State Machine)^[4]是对状态行为建模的最直观的方法, 也是实现事件驱动系统的一种很好的形式方法。主要用来描述对象、子系统、系统的生命周期。通过层次状态机可以了解到一个对象能到达的所有状态以及对象收到的事件对对象状态的影响等。状态机指定对象的行为以及不同状态行为的差异。同时, 它还能说明事件是如何改变一个对象的状态。因此非常适用于嵌入式软件开发。为便于理解, 下面给出 HSM 中的相关概念。

状态: 状态是对象的生命周期中满足某种条件, 执行某些动作或等待某些事件发生的一

个阶段。

事件：事件是一个在时空中显示出现的特定现象，它可以触发状态转换。

转换：转换是从一个状态结点到另一个状态结点的移动。

层次状态机（如图 1 所示）较之经典的状态机，最重要的改进就是引入了层次式状态。状态层次嵌套的主要特性来自抽象与层次的结合。这是一种降低复杂性的传统途径，也就是软件中的继承。在面向对象^[3]中，类继承概念描述了类和对象之间的关系。类继承描述在类中的 is-a 关系。在嵌套状态中，只需用 is-in 状态关系代替 is-a 类关系，即它们是等同的分类法。状态嵌套允许子状态继承来自其超状态的状态行为，因此它被称为行为继承。在实现中行为继承有意的把自己仅限于被动事件处理器，即它处理事件需要外部的驱动。而不包括传统上与状态机有关的标准元素如：事件排队、事件分发器、执行上下文（线程）或定时服务等。因此可以把这些与应用领域和操作系统相关的部分抽象出来，构建了一个可复用的适合嵌入式软件开发领域的框架。

2.2 应用框架 State Machine Wizard 的设计

一个基于层次状态机的应用程序对象的执行环境，除了提供事件处理器外，还必须提供执行上下文线程、事件排队、事件分发器和定时服务等^[1]。这些元素强烈依赖与应用领域和操作系统的的功能支持。在一个特定领域内如：嵌入式软件开发领域中，它们在各个系统中的改变很小，因此能在很多应用中被重复使用。基于这种思想可以把这些元素抽象出来，构建成适合嵌入式软件开发领域的可复用的应用框架。和其他应用框架一样，该应用框架不包括构件应用程序的小片程序，而是实现嵌入式软件开发领域中通用完备功能的底层服务。使用该应用框架的编程人员可以在一个通用功能已实现的基础上开始具体的系统开发。

通过以上分析，可以把嵌入式软件系统的各成份，按照“最大独立”原则^[2]分成三层（如图2所示），以分层的形式来组织系统，并确定层内和层间的联系方式。并且要求每层向它的上层提供服务；同时向它的下层请求服务。

硬件抽象层：是一个实时操作系统（RTOS），为多线程奠定基础并提供类似事件队列和内存池的基本服务。

公共业务层：既是我们将要构建的应用框架层，它把业务域中绝大多数应用任务需要的公共功能（主要是事件排队、事件分发器和定时服务等）抽象为公共的业务对象，封装业务域中绝大多数应用任务的公共数据。并且为具体业务层提供丰富的API 接口。

具体业务层：既是基于该应用框架构建的应用程序，也就是用户依据所要求实现的具体任务去开发的软件包。

为了满足这种设计的需要，把具体业务层分解为若干个独立的应用程序对象。为了与 UML 规范相一致，在此我们把该应用程序对象叫做活动对象，同时每一个活动对象以一个状态机的方式运行。它们之间并发执行并通过发送和接收消息来进行通信。为了把公共业务层抽象成一个可复用的适合嵌入式软件开发领域的框架，该层必须提供一个基于状态机的应用程序所需的事件排队、事件分发器和定时服务等。在下面就该框架设计中的关键技术进行讨论。

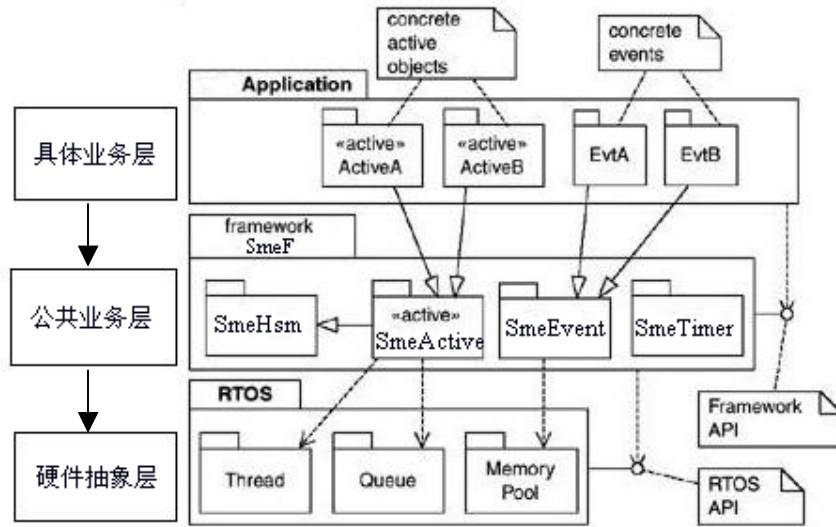


图 2：嵌入式实时系统设计的三层结构图

2.2.1 State Machine Wizard 的数据结构

在此定义了一系列特殊的宏作为状态机的映射数据。这些宏映射到状态枚举、事件处理函数声明、一个状态的事件处理函数表、状态树定义和应用程序变量定义。具体如下定义。

状态枚举：

```
#define SME_BEGIN_STATE_DECLARE(_app) enum _app##_state_enum_t \
{#define SME_STATE_DECLARE(_state) _state, #define SME_MAX_STATE(_app) \
_app##_max_state #define SME_END_STATE_DECLARE };
```

状态事件处理函数表定义：

```
#define SME_ENTRY_FUNC_IDX 0
#define SME_EXIT_FUNC_IDX 1
#define SME_EVENT_HANDLER_FUNC_IDX 2
#define SME_BEGIN_STATE_DEF(_app, _state) \
static const SME_EVENT_TABLE_T _app##_state##_event_hdl_tbl[] = \
{#define SME_STATE_ENTRY_FUNC( _EntryFunc) \
{ SME_INVALID_EVENT_ID, _EntryFunc, 0}, \
#define SME_STATE_EXIT_FUNC( _ExitFunc) \
{ SME_INVALID_EVENT_ID, _ExitFunc, 0}, \
#define SME_ON_EVENT( _EventID, _Handler, _NewState) \
{ _EventID, _Handler, _NewState}, \
#define SME_END_STATE_DEF { SME_INVALID_EVENT_ID, 0, SME_INVALID_STATE} \
};
```

状态树定义：

```
#define SME_BEGIN_STATE_TREE_DEF(_app) \
extern const SME_STATE_TREE_TABLE_T _app##_state_tree[] = \
{#define SME_STATE(_app, _state, _state_parent, _def_substate) \
{(SME_EVENT_TABLE_T*)_app##_state##_event_hdl_tbl, \
_state, _state_parent, _def_substate}, \
#define SME_END_STATE_TREE_DEF };
```

应用程序定义：

```
#define SME_APPLICATION_DEF(_app, _app_name) \
```

```

struct SME_APP_T _app##App = { \
    _app_name, NULL, NULL, 0, NULL, NULL, _app##_state_tree, SME_INVALID_STATE};
#define SME_GET_APP_VAR(_app) _app##App
#define SME_DEC_EXT_APP_VAR(_app) extern SME_APP_T _app##App;

```

2.2.2 State Machine Wizard 的类型定义

状态定义：

```

typedef short SME_STATE_T;
#define SME_APP_STATE 0
#define SME_INVALID_STATE -1

```

事件定义：其中正 32 位整数值代表用户定义的事件标识符，负 32 位整数值代表 State Machine Wizard 定义的事件标识符

```

typedef long SME_EVENT_ID_T;
#define SME_INVALID_EVENT_ID -1
#define SME_EVENT_KILL_FOCUS -2
#define SME_EVENT_SET_FOCUS -3
typedef enum
{
    SME_EVENT_CAT_UI=0,
    SME_EVENT_CAT_OTHER,
};
typedef unsigned char SME_EVENT_CAT_T;
typedef enum
{
    SME_EVENT_ORIGIN_INTERNAL=0,
    SME_EVENT_ORIGIN_EXTERNAL,
};
typedef unsigned char SME_EVENT_ORIGIN_T;
typedef enum
{
    SME_EVENT_DATA_FORMAT_INT=0,
    SME_EVENT_DATA_FORMAT_PTR,
};
...

```

2.2.3 事件派送

上面定义了 State Machine Wizard 设计中用到的数据结构和类型定义。由于活动对象是通过事件发送和接收来进行通信的。因此事件传送是整个 State Machine Wizard 的核心也是最复杂的方面。为了准确的实现该框架的功能，事件传送机制应包括动态事件分配、事件订阅、事件出版、事件组播以及自动事件回收。这些功能的具体实现通过事件传送函数 SmeDispatchEventv() 来完成。该函数描述如下：

```

BOOL SmeDispatchEvent(struct SME_EVENT_T *pEvent, struct SME_APP_T *pApp)
{
    SME_STATE_T nOldState; /* Old state should be a leaf.*/
    SME_STATE_T nState;
    SME_STATE_T nNewState;
    int i;
    BOOL bFoundHandler=FALSE;
    SME_EVENT_HANDLER_T pHandler = NULL;

```

```

SME_STATE_T OldStateStack[SME_MAX_STATE_TREE_DEPTH];
SME_STATE_T NewStateStack[SME_MAX_STATE_TREE_DEPTH];
SME_STATE_T nOldStateStackTop =0;
SME_STATE_T nNewStateStackTop =0;

struct SME_EVENT_TABLE_T *pStateEventTable;

if (pEvent==NULL || pApp==NULL) return FALSE;

nOldState = pApp->nAppState; /* Old state should be a leaf. */
nState = nOldState;
    pStateEventTable = pApp->pStateTree[nOldState].pEventTable;
    ...
    ...
    ...
}

```

以上介绍了应用框架实现时的关键问题。当然，在框架的具体实现中还有很多方面要考虑。比如：内存管理、处理错误和异常的条件、互斥和阻塞等等。从图 2 可知，该三层结构的设计有效的把应用框架层与最下面的硬件抽象层分离开来，应用程序对象通过框架提供的 API 来使用框架提供的各种服务，如：事件的传送和定时服务等。应用程序对象通常不需要直接访问硬件抽象层提供的 API，这样当改变框架的硬件抽象层时，只需移植框架代码而不会影响到基于该应用框架的上层应用程序。从而使得该应用框架更易于复用。

图 3 是嵌入式系统结构图，嵌入式系统应用程序运行于一独立的线程，它可于调用其它模块提供的服务或接受其它模块发送过来的消息，这些提供服务的模块运行于其它独立的线程。整个系统可先运行于 Windows 的仿真环境或迁移到目标环境中。

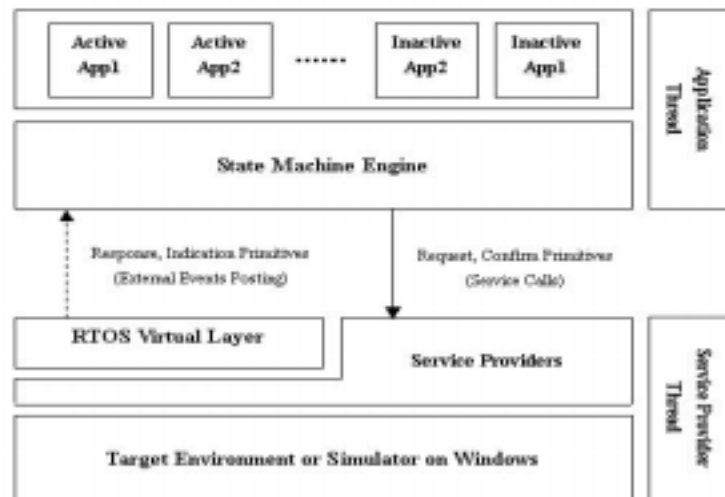


图 3：嵌入式系统结构图

3 实现与应用

根据上面的设计思路，我们实现了一个在嵌入式软件开发领域中可复用的应用框架 State Machine Wizard 简称 SMW。SMW 在实际应用中所处的位置如图 3 所示。只要遵循该应用框架定义的标准，基于该应用框架可以开发适合各领域的嵌入式软件。活动对象通过使用框架提供的 API 来使用该框架的各种服务。

我们在遵循该应用框架定义标准的基础上，使用框架定义的一系列 API 函数开发了一个具

体的手机嵌入式系统。由于使用了该框架，开发周期大大缩短。并且在具体应用过程中，针对图 2 所示的层次结构，配套开发了一些代码生成工具，减少了大量的重复编码工作，项目开发效率明显提高。如：帮助自动生成活动对象中的重复代码部分的工具和帮助生成资源代码文件的工具。图 4, 5 是利用工具 UML State Machine Wizard^[6]开发具体应用程序时，建模阶段的状态图和仿真运行阶段的状态树。图 6 展示了利用该引擎和它的配套工具开发应用程序的全过程。

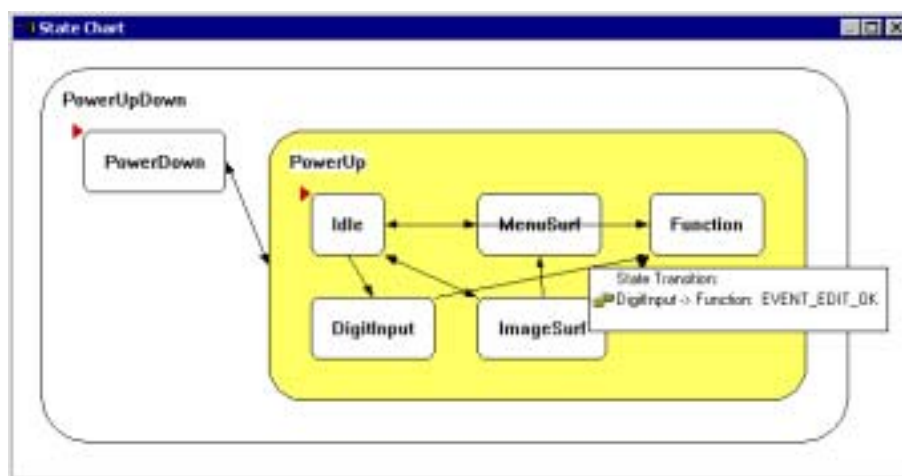


图4：建模阶段的状态图

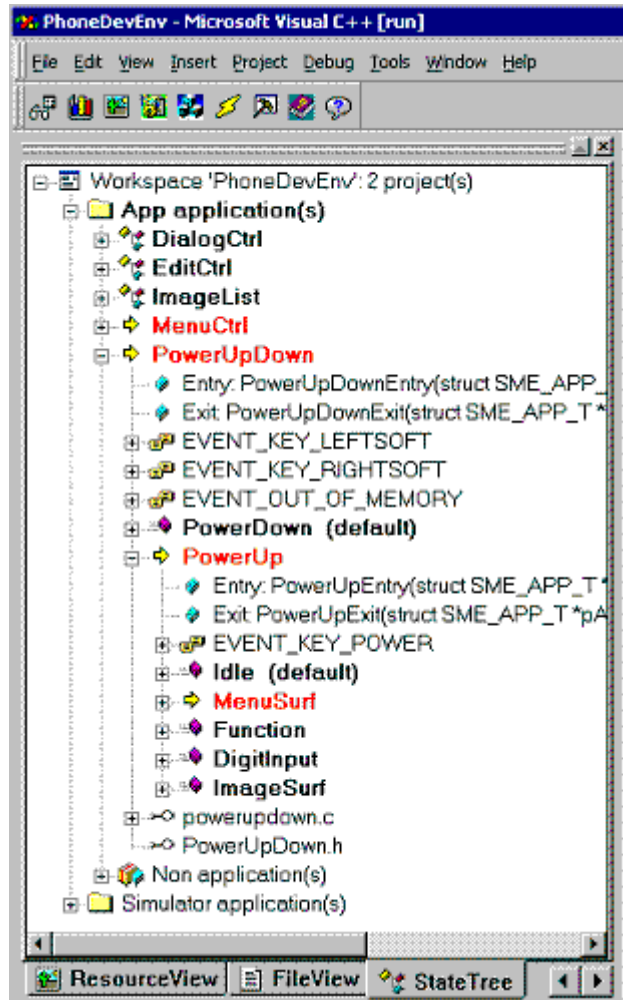


图 5：仿真运行阶段的状态树

在具体应用中，该应用框架至少在3个方面达到了预期目标：1) 模块化，该框架将多变的实现细节封装于固定的接口之中，提高了软件的模块性；2) 可复用性，该框架提供的固定接口被定义类属组件，并可以用来构造新的应用程序；3) 可扩展性，该框架通过提供显式的钩子方 (HookMethods^[1])，允许应用程序来扩展其固定接口。

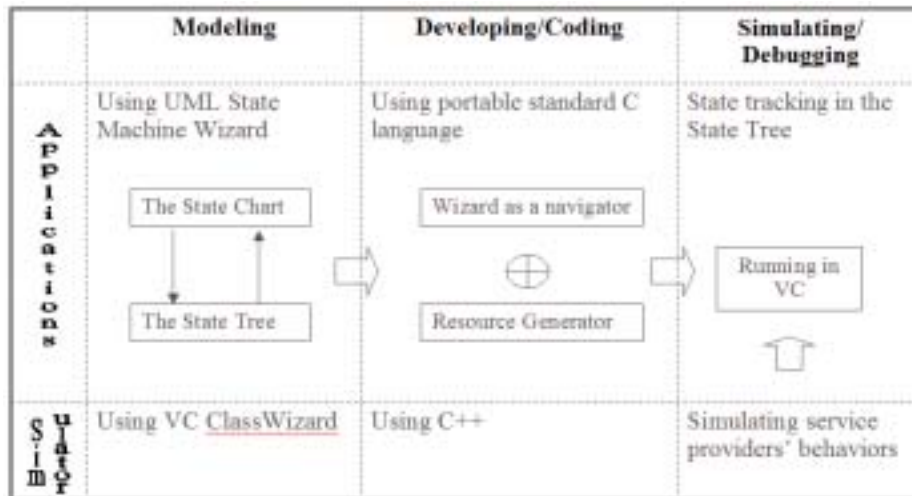


图 5：开发过程

4 总结

应用框架的设计即系统体系结构的设计，有助于实现领域内体系结构层次较大粒度的设计复用。本文首先对层次状态机本质和嵌入式软件系统的层次结构进行系统化的分析，然后利用应用框架技术把嵌入式软件开发领域中与应用领域和操作系统相关的部分抽象出来，构建一个可复用的适合该领域的应用框架。该框架提供了一套可复用的 API 接口，使得领域开发人员只需关心基于该框架的应用程序的开发，从而大大得缩短了开发周期。而且在实现该框架时我们开发的一些代码自动生成工具，大大减轻了重复代码编写的工作量，提高了开发效率。

该软件 UML State Machine Wizard 可在 www.intelliwizard.com 下载并获得更多资料。

参考文献：

- [1] Samek M. Practical Statecharts in C/C++: Quantum Programming for Embedded Systems [M]. CMP BOOKS, 2002.
- [2] Barr, Michael. Programming Embedded Systems in C and C++. Oreilly & Associates[M], 1999
- [3] Duby, Carolyn. Class 203: Implementing UML statechart diagrams. Proceedings of Embedded Systems Conference[C], Fall, San Francisco, 2001
- [4] Object Management Group, Inc. OMG Unified Modeling Language Specification v1.4. <http://www.omg.org>, 2001, September
- [5] Samek, Miro, Paul Y Montgomery. State-Oriented Programming. Embedded Systems Programming, 2002, August:22-23
- [6] UML State Machine Wizard[Z]. <http://www.intelliwizard.com/>

UML State Machine Wizard

(Department of Computer Science and Engineering, Southeast University, Nanjing 210096)
He Wei, Jin Yuanping

Abstract: With the advancement of software reusable technology, reusable application framework is becoming more and more important. Application framework will contribute to the implementation of reusable domain architecture, which will be the kernel of a software system. This paper studies the hierarchic architecture of software development in embedded system based on HSM. Finally, we will give an implementation of a valuable reusable application framework State Machine Wizard.

Key words: Application framework; HSM; Active object; state-oriented; UML