计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(8):83−87 [doi: 10.15888/j.cnki.csa.005903]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于模型驱动的实时嵌入式系统①

赵　勇, 陈香兰

(中国科学技术大学 计算机学院, 合肥 230026)

摘　要: 随着实时嵌入式系统的功能越来越复杂, 现有的软硬件分离、软硬件协调等实时系统设计方法已经无法满足其系统实现的要求. 本文根据模型驱动开发架构 MDA 和模型集成开发 MIC 的核心思想, 将时间语义结合服务体/执行流(Servant/Exe-Flow Model, 简称 SEFM)模型, 提出了一种基于模型驱动的实时系统设计方法. 首先, 本文给出了 SEFM 模型的元模型表达系统的抽象语义, 同时使用 XML 语言和框图语言来描述 SEFM 模型的具体语法. 结合 XML 解析技术, 根据同一抽象语法的不同具体语法能够相互转化, 实现了框图语言的代码生成, 最后以实时跟车系统设计方案表明该系统实现方法的可行性和正确性.

关键词: 实时; 嵌入式; MDA; MIC; SEFM

引用格式: 赵勇,陈香兰.基于模型驱动的实时嵌入式系统.计算机系统应用,2017,26(8):83–87. http://www.c-s-a.org.cn/1003-3254/5903.html

Real-Time Embedded System Design Method Based on Model-Driven

ZHAO Yong, CHEN Xiang-Lan

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China)

Abstract: As the real-time embedded systems are more and more complex, the existing RTOS design method, such as hardware and software separation ,hardware and software coordination and so on, is unable to meet the requirements of its implementation. Combined with the core idea of MDA and MIC, this paper proposes a method based on model-driven for the RTOS design , which combines the temporal semantics with the Servant / Exe-Flow Model. Firstly, the paper gives the abstract semantics of the meta model expressing SEFM , and describes the concrete syntax of SEFM using XML language and block diagram language. If a different specific syntax can express the same abstract syntax, then each of them can be transformed into the other. Combined with XML parsing technology, the code generation of SEFM can be realized. Finally, the experiments of the following vehicle system show that the method of system design is feasible and correct.

Key words: real-time; embedded system; MDA; MIC; SEFM

1　概述

嵌入式系统是以应用为中心, 以计算机技术为基础, 软硬件可裁剪, 适应应用系统对功能、功耗、可靠性、体积、成本严格要求的专用计算机系统[1]. 当前嵌入式系统已被广泛地应用到航天航空、工业控制、消费类电子产品等领域, 嵌入式系统设计也变得越来越复杂, 一般会涉及硬件与软件, 以及功能和时间等多个方面的要求.

传统的嵌入式系统设计方法已经无法满足复杂嵌入式系统的设计需求了, 主要表现在以下几个方面: (1)软硬件分离的设计方法, 只能实现软硬件性能的各自优化, 无法达到系统的整体性能最优. (2)软硬件协同设计方法对系统开发者提出了很高的要求——对系统统的软件和硬件熟悉程度到了苛刻的要求. (3)嵌入式

① 基金项目: 国家自然科学基金(61379040, 61272131, 61202053); 江苏省自然科学基金(SBK2012194)
收稿时间: 2016-12-05; 采用时间: 2017-01-04

系统不仅功能变得越来越复杂, 而且对系统完成这些功能的响应时间也提出了更为严格的要求.

近年来, 随着软件工程和系统模型技术的发展, 模型驱动开发的系统设计理念被越来越多的系统开发人员认可和使用. 模型驱动系统软件开发是对现实世界建立模型、转换模型, 直到生成可执行代码. 在模型驱动系统开发过程中, 模型是软件开发的核心. 软件的开发和更新过程就是以模型为载体, 通过模型之间的映射机制来驱动的过程, 也可以认为是高抽象级模型向低抽象级模型的逐层转化和实现的过程.

2002 年, 对象管理组织 OMG(object management group)提出了模型驱动架构 MDA(Model Driven Architecture). MDA 结合面向对象技术, 利用统一建模语言 UML 对系统进行分析建模, 再通过模型得到代码. MDA 提倡将模型作为核心, 贯穿于系统设计的整个开发周期, 用模型描述系统的需求、设计、测试、维护等过程. 但是, MDA 作为一个通用的软件开发标准, 缺乏面向领域开发的支持. 范德堡大学的 Janos Sztipanovits 和 Gabor Karsai 提出了模型集成计算 MIC (Model-Integrated Computing). MIC 是一种以模型为中心的软件系统开发理论, 其模型分为 4 个层次, 从上到下依次为元元模型层、元模型层、模型层以及实现层. 在嵌入式领域中, MIC 已经得到广泛关注和应用. 目前, MIC 在嵌入式系统的模型研究更多地局限于功能的设计, 而缺乏时间语义的支持.

本文在深入研究 MDA 和 MIC 方法的基础上, 将时间语义结合 SEFM 模型, 提出了一种基于模型驱动架构的实时嵌入式系统设计方法, 重点表述了层次间模型的映射转化和代码生成的设计, 最后以跟车系统案例建模仿真实现加以说明本设计方法的可行性和有效性.

2 层次的系统模型概述

前面提到, 传统的嵌入式实时系统设计的不足之处是, 高层次的抽象阶段与目标环境中的开发编程阶段脱节. 这是因为各个抽象层次之间的映射关系不明确, 高层次的应用抽象描述无法直接映射成抽象模型, 抽象模型也无法直接转化成代码. 实际的系统设计还需要手动编写代码, 系统的功能特性和时间特性都是经过代码的某种组合才能完成, 这个过程是艰难而容易出错的.

本文将嵌入式系统按照设计阶段分为 5 个层次. 如图 1 所示, 从上到下依次是应用描述层, 抽象模型层, 编程语言层, 可执行代码层, 硬件层. 模型驱动开发关键在于最上面的三个层次的研究, 应用描述层位于整个开发过程的最顶层, 它定义了模型的建模元素, 一般是一种领域无关的通用模型描述语言. 该层不仅需要描述整个系统的行为, 而且需要分析和验证系统, 为后续开发打下基础. 抽象模型层采用领域内抽象模型对嵌入式实时系统进行抽象描述和刻画. 利用仿真平台对整个系统或局部系统进行仿真, 并观察仿真结果. 当系统满足设计要求时, 将领域抽象模型变换成代码. 最后, 结合硬件平台代码, 使用嵌入式集成开发工具, 将生成的可执行软件代码移植到相关的硬件平台, 完成系统实现.

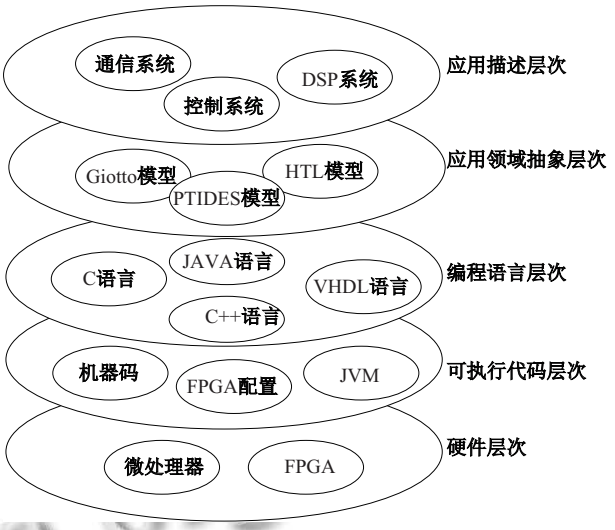


图 1 嵌入式系统设计层次图

在实时嵌入式领域中, 系统行为的正确性不仅取决于计算的逻辑结果, 而且与产生这些结果的物理时间有关. 传统的实时系统模型无法保障任务释放时间的确定性, 尽管可以通过反复测试使得系统具有可靠性, 但这种方法存在缺陷, 其根本原因在于: 实时系统模型中缺乏明确描述时间属性的语义.

中国科学技术大学龚育昌教授等人提出了执行流/服务体(SEFM)模型, 并在此基础上进行了时间可预测实时模型的深入研究. 执行流是对 CPU 执行能力的抽象, 每个 CPU 对应一个执行流抽象; 服务(Service)是最基本的功能单元, 也是调度的单位, 任务由一组服务组合而成, 服务描述了任务间的交互方式以及各任务的时间属性.

3 SEFM 元模型的设计与实现

根据 MDA 和 MIC 方法提出的元模型思想, 可使用元模型刻画 SEFM 最核心的语义, 也就是 SEFM 的抽象语法. 在元模型的基础上再实现时间约束和服务体组件刻画, 可实现多种语言来描述 SEFM 的具体语法. 这样基于相同抽象语法的具体语法的不用语言可以实现相互转化.

3.1 SEFM 的抽象语法

编程语言的语法和语义有着很重要的区别, 语法为构造有效语言制定规则, 语义则表示语言的含义. 模型也有着类似的区别, 语法为模型如何表示制定规则, 语义则表示模型意味着什么含义.

模型的抽象语法代表着模型的结构, 比如模型可以使用图的结构表示, 由点和边构成, 边连接着点. 或者使用树结构表示, 可以定义层次结构. 相比较而言, 模型的具体语法表达抽象语法的特殊符号, 如框图语言. 具体语法代表的所有结构集合能够用来表示模型的抽象语法. 也就是抽象语法包含了模型的抽象结构, 具体语法提供文本或者图描述模型.

抽象语法比具体语法更加基础, 如果针对同一个抽象语法表达的两个具体语法, 那么一个转化成另一个往往是容易的. 通常, 元模型是一种模型的建模语言或符号. 在建模过程中, 工程师经常使用元模型来精确定义抽象语法. 元模型形式化地定义了语言的各种组成元素以及它们的抽象语法. 在 MDA 和 MIC 中, 元模型一般使用 UML 类图表示. SEFM 模型的 UML 元模型如图 2.

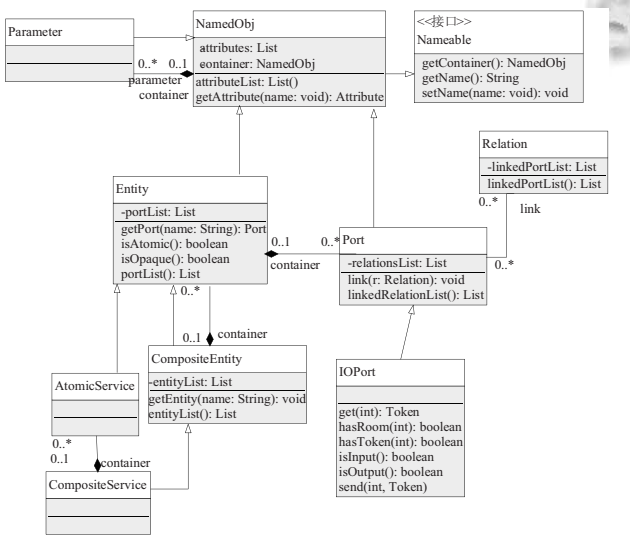


图 2 SEFM 的元模型

SEFM 模型中的每个组件都是 NamedObj 类的实例. NamedObj 有四个子类, 分别是属性 Parameter、实体 Entities、端口 Port 和关系 Relation. SEFM 模型由顶层实例包含其他实例组成, 这些实例通过端口进行交互. 所有的对象(实例, 端口, 关系)都可以分配属性, 用来定义参数或者注释. 端口间通过链接来确定关系, 表示元模型中关系类和端口类之间的关联关系.

3.2 SEFM 模型的具体语法

上文提到, 具体语法是抽象语法的更高层次描述. 抽象语法是定义数据结构来表示模型, 具体语法则是捕获如何呈现机器间通信或与人类交互. 具体语法一般可使用编程语言表示, 常用的替代具体语法的语言是可扩展标记语言 XML. 从常见的元模型可以推导出建模语言的可视化语法, 也可以是文档类型定义或 XML 文档. 每个元素的图形语言将对应于一个 XML 文件元素. 比如使用图形符号的输入端口 IP0 对应 XML 文件元素<inputport name=IP0>.

区别模型的抽象语法和具体语法对于建模技术会产生一个深远的影响: 编辑和操作. 建模者与具体语法进行交互: 使用具体语法的“原始”概念来创建和修改模型结构. 如果开发环境是图形化的, 那么开发人员可以直接对图形对象进行操作. 这些图形对象只是底层抽象语法对象的渲染; 因此, 图形对象的变化将导致底层对象的更改.

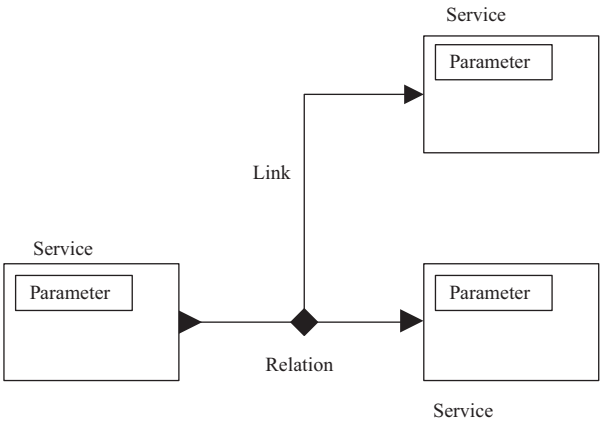
3.3 代码自动生成

随着高层次抽象模型的发展, 基于模型的代码生成技术(Model-Based Code Generation, 简称 MBDG)成为一个新的研究领域. 代码生成的核心思想就是将模型的建模语言转化成可执行语言. 代码生成能够从设计的模型中有效的综合实现代码. 理想情况下, 在系统的设计阶段, 代码生成可以实现模型在不同平台下的实现.

生成编码社区(GPCE)着重推广了一种基于模板的元编程技术, 编译器工具如 AspectJ 和 AHEAD 为这种编程范例提供标准, 涉及模板语言、源建模语言的元模型的使用和目标语言和平台的描述. 基于模板的元编程技术采用一种编程语言操纵其他编程语言完成代码生成. 操纵语言称为元语言, 被操纵语言称为对象语言. 对象语言使用模板作为语言的第一对象数据, 在编译时模板会被解释为平台相关的可执行语言. 通过对基于模板的元编程技术研究, 本文模型的代码生成思想如图 4 所示, 代码生成过程可以等价为一个模型转

换, 它为模型生成其他平台语言提供了规范, 呈现相同模型的不同语法实现, 同时保留了模型的抽象语义.

实时嵌入式系统一般使用 C 语言编写功能代码, 本文也采用 C 语言作为对象语言, 也就是模板. C 语言模板是一个.c 文件, 这样可以保证文件的结构唯一. 适配器是代码生成框架的关键抽象, 每个模型组件都与一个适配器. 为了实现可读性和可维护性的适配器, 目标代码块的适配器被放置在同一目录下的不同文件中.



(a) SEFM 模型的框图语法

```
01  <?xml version="1.0" standalone="no"?>
02  <entity name="TopLevel" class="Entity">
03  <property name="Attribute" class="Attribute"/>
04    <entity name="A" class="Entity">
05      <property name="Attribute" class="Attribute"/>
06      <port name="p" class="IOPort"/>
07      <relation name="r" class="IORelation"/>
08      <link port="p" relation="r"/>
09    </entity>
10    <entity name="B" class="Entity">
11      <port name="r" class="IOPort"/>
12    </entity>
13    <entity name="C" class="Entity">
14      <property name="Attribute" class="Attribute"/>
15      <port name="q" class="IOPort"/>
16      <relation name="r" class="IORelation"/>
17      <link port="q" relation="r"/>
18    </entity>
19    <relation name="r" class="IORelation"/>
20    <link port="A.p" relation="r"/>
21    <link port="B.r" relation="r"/>
22    <link port="C.q" relation="r"/>
23  </entity>
```

(b) SEFM 模型的 XML 语法

图 3 SEFM 模型的框图语法和 XML 语法

对于每个适配器来说, 对象代码包含的代码模板文件都是人工手写的, 并且已经验证了代码的正确性(即语言功能上等价于 SEFM 模型组件). 代码模型存储在代码库中, 针对相同模型的代码生成来说, 代码模型是可重用的. 手动编写模板还保留了生成代码的可读性, 同时使用宏处理组件实例的具有信息(如端口信息, 参数变量等).

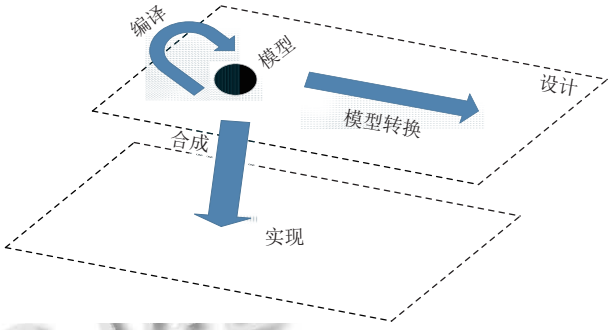


图 4 代码生成思想

3.4 实现框架

本文提出的模型系统设计方法是面向框图语言、并行计算和仿真等技术在嵌入式系统设计中的综合应用, 该方法将模型驱动架构的核心思想贯穿整个系统设计的全过程.

如图 5 所示, 是基于模型驱动的实时嵌入式系统方法实现框架. 在应用问题层使用框图语言来描述系统的功能属性, 使用逻辑时间(时间戳)的语义来描述系统的时间属性. 使用时间戳的时间语义可以有效的支持系统各个簇之间的事件同步和时间同步. 通过建模可以得到仿真的结果, 如果仿真结果能够达到预期目标, 利用代码生成技术, 就可以生成相应的功能代码了. 结合 WCET 和调度器算法分析功能代码的可调度性. 如果产生的功能代码, 不满足可调度性, 就需要重新建模, 直到产生的功能代码满足可调度性. 将功能代码和系统代码链接在一起, 通过编译器写入到相关硬件.

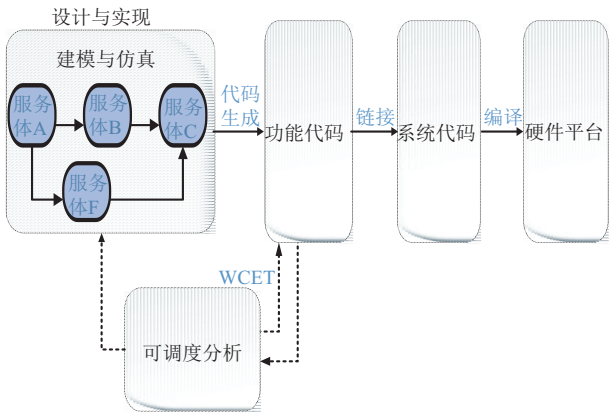
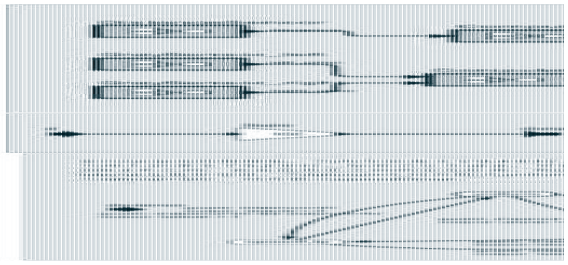


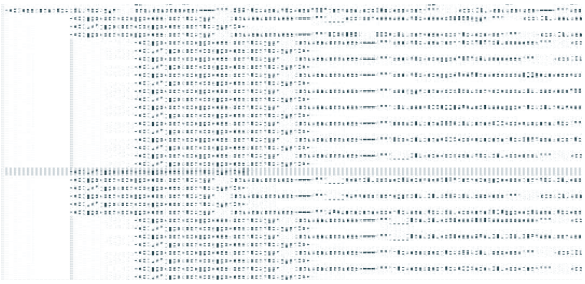
图 5 模型系统的实现框架

4 实验与总结

本文用两种方案实现了实时跟车系统: 1)使用 SEFM 模型实现, 如图 6 所示; 2)在μC OS 上进行实现. 响应时间是衡量实时系统性能的一个重要指标, 表 1 显示了不同方案中任务实际响应时间与实际需求时间的统计结果. 从实验结果可以看出 SEFM 能够较好的满足实际的时间需求, 波动较小. 其主要原因是因为模型驱动在系统设计整个过程始终将功能和时间紧密的结合, 确保模型映射和转换过程中, 时间语义不发生改变, 同时, 也不随着硬件平台的变化而变化.



(a)实时跟车系统建模



(b)模型的部分XML代码

图 6 跟车系统建模与实现

表 1 不同实现方案的模式响应时间

实现方式	加速	减速	左转	右转
功能需求	20 ms	20 ms	10 ms	10 ms
μC	18.4 ms	17 ms	10.3 ms	9.8 ms
SEFM	20.1 ms	20.0 ms	9.9 ms	10.0 ms

本文结合 MDA 和 MIC 的模型驱动思想, 将时间语义结合 SEFM 模型, 提出了一种基于模型驱动架构的实时嵌入式系统设计方法. 将系统的功能设计和时间设计贯穿整个实时系统实现的全部过程. 使用元模型来表达 SEFM 的抽象语义, XML 语义和框图语言来表达 SEFM 的具体语言, 这样的设计可以确保系统在转化过程中语义的不变. 通过实验发现, 这样的设计能够确保系统在实际运行时的物理时间满足系统需求. 目前, 建模平台功能比较单一, 需要进一步完善建模平台的建模仿真功能.

参考文献

1 Schirner G, Götz M, Rettberg A, *et al*. Embedded systems: Design, analysis and verification. Berlin Heidelberg: Springer, 2013.

2 Henzinger TA, Sifakis J. The embedded systems design challenge. International Symposium on Formal Methods. Berlin Heidelberg, Germany. 2006. 1–15.

3 Miller J, Mukerji J, Belaunde M. MDA Guide V1.0.1. Object Management Group, 2003.

4 Völter M, Stahl T, Bettin J, *et al*. Model-driven software development: Technology, engineering, management. New York: John Wiley & Sons, 2013.

5 Sztipanovits J, Karsai G. Model-integrated computing. Computer, 1997, 30(4): 110–111. [doi: 10.1109/2.585163]

6 Iacovella CR, Varga G, Sallai J, *et al*. A model-integrated computing approach to nanomaterials simulation. Theoretical Chemistry Accounts, 2013, 132: 1315. [doi: 10.1007/s00214-012-1315-7]

7 Atkinson C, Kühne T. Model-driven development: A metamodeling foundation. IEEE Software, 2003, 20(5): 36–41. [doi: 10.1109/MS.2003.1231149]

8 Lee I, Leung JYT, Son SH. Handbook of real-time and embedded systems. Florida: CRC Press, 2007.

9 Krahn H, Rumpe B, Völkel S. Roles in software development using domain specific modeling languages. Proc. of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM‘ 06). Portland, Oregon, USA, 2014.

10 龚育昌, 张晔, 李曦, 等. 一种新型的构件化操作系统的内核设计. 小型微型计算机系统, 2009, 30(1): 1–7.

11 吴明桥, 陈香兰, 张晔, 等. 一种基于服务体/执行流的新型操作系统构造模型. 中国科学技术大学学报, 2006, 36(2): 230–236.

12 Zhou Y, Lee EA. Causality interfaces for actor networks. ACM Trans. Embedded Computing Systems (TECS), 2008, 7(3): 29.

13 Jensen JC. Elements of model-based design. University of California, Berkeley, Technical Memorandum. UCB/EECS-2010-19, 2010.

14 Ptolemaeus C. System design, modeling, and simulation: Using ptolemy II. Berkeley: Ptolemy, 2014.

15 Marwedel P, Goossens G. Code generation for embedded processors. US: Springer Science & Business Media, 2002.

16 Rajamani R, Choi SB, Law BK, *et al*. Design and experimental implementation of longitudinal control for a platoon of automated vehicles. Journal of Dynamic Systems, Measurement, and Control, 2000, 122(3): 470–476. [doi: 10.1115/1.1286682]