第33卷 第21期 计 算 机 工 程 2007年11月
Vol.33 No.21 Computer Engineering November 2007
·软件技术与数据库· 文章编号：1000—3428(2007)21—0055—03 文献标识码：A 中图分类号：TP391

# 非精确实时计算的方面框架及编织测试

申利民，徐富林
(燕山大学信息工程学院，秦皇岛 066004)

**摘 要：**针对应用面向方面技术提出了一个非精确实时计算的框架。该框架允许开发人员将可选任务定义和设计为方面，基于运行时可用资源动态编织可选的方面。通过一系列编织与解编织的时间测试，找到影响编织与解编织时间的两个主要因素——通知和目标方法的数目。测试结果表明，在实时系统中动态编织技术可实现非精确计算的可行性，且无需引入不可预测性。

**关键词：**面向方面；实时；非精确计算；动态编织；编织测试

# Aspect-oriented Framework and Weaving Test for Imprecise Real-time Computation

SHEN Li-min, XU Fu-lin
(College of Information Science and Engineering, Yanshan University, Qinhuangdao 066004)

**【Abstract】** Aspect-oriented technology is applied to imprecise computation framework which allows real-time system developers to define and design optional task as aspects. At running time, the optional tasks are dynamically woven or unwoven based on current available resources. A set of tests of weaving and unweaving aspects for the framework are carried out. The tests found that two major factors which influence weaving and unweaving time are the number of advices and the number of targeted methods. Results show the feasibility of using dynamic weaving to achieve imprecise computation in real-time computation. the dynamic weaving has not introduced the time unpredictability into real-time applications.

**【Key words】** aspect-oriented; real-time; imprecise computation; dynamic weaving; weaving test

传统的实时系统为满足其时间约束条件，通常按最坏情况配备资源。这种方法导致在正常情况下系统资源得不到充分利用，而一旦按正常情况配备资源时，在最坏情况下的执行时间要比正常情况下的执行时间超出很多。在实时系统中引入非精确计算，既可以满足系统对时间期限的要求又可提高资源利用率[1]。在非精确实时系统中，可将实时系统的任务按对时间的要求分为强制执行部分和可选执行部分。强制执行部分必须在规定时间内完成，可选执行部分则可根据系统状态有选择地执行。

当前对非精确计算问题的研究主要集中于调度方法上[2~4]，这些方法通常需要对所有任务进行集中调度，而很难满足动态性需求。对系统开发人员来说，描述和自定义调度策略，并基于需要动态的加载可选执行部分，非常困难。因此，本文提出了一个框架，采用面向方面技术(aspect-oriented programming, AOP)，将可选部分设计成方面，并根据运行时系统状态决定哪些可选部分被执行。其优点在于：不需直接修改底层代码就可动态添加可选执行计算，使可选执行任务的表示更加简单，并且能提高软件的重用性和模块性。

目前，在AOP方面有许多可利用的工具[6]，如AspectWerkz。但在实时系统中使用面向方面技术的可行性上，测试工作做得很少[5~6]。因此，本文设计了一种测试方法，用于检验影响方面编织时间的相关参数，并分析其对实时系统时间可预测性的影响。

## 1 框架设计与实现

### 1.1 框架设计

框架的构建可以使开发人员基于框架对非精确实时系统的任务进行定义和设计。将每个完整的任务划分为一个强制执行部分以及若干个可选执行部分。强制部分必须执行且不可拖延，可选部分被定义为方面，由框架控制和协调各个方面的动态执行，使系统在满足时间约束的条件下尽可能多地执行可选任务并得到尽可能精确的结果。当系统资源紧张时，框架可保证强制任务的执行，这样系统也能得到一个可用的结果。而当资源充足时，框架可以根据系统状态，决定哪些可选部分被执行，从而使结果更加精确。并且通过使用面向方面技术，可在不修改现有系统的前提下，引入新的可选执行计算，添加新的功能性并提升现有任务的执行效果。

为简化原型又保证其一般性，假定：强制执行部分可以被抽象到一个单独的主方法中。框架必须提供一个机制，用以动态地编织可选执行部分。由于可能存在多个可选部分，因此框架必须同时提供一个决策机制来决定执行哪些可选部分。图1描述了动态编织的非精确计算模型。

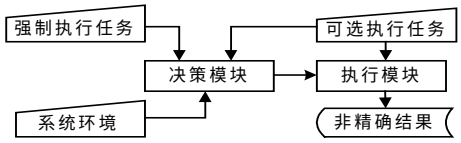


**图1 动态编织的非精确计算模型**

在每次完成强制任务后，决策机制被激活并判断剩余资源是否足够用于进行可选任务的选择。如资源充足，决策机制进一步决定所有可选任务中哪一个能被安全地执行。结果

**基金项目：**国家留学基金资助项目(2003813003)
**作者简介：**申利民(1962－)，男，博士、教授、博士生导师，主研方向：柔性软件工程，软件协同技术；徐富林，硕士
**收稿日期：**2007-03-19 **E-mail：**shenllmm@sina.com

信息被传递给运行机制，对选中的任务编织和执行，待完成后解除编织，减少资源占用。决策模块需采集系统环境信息，如 CPU 的占用情况，进行决策过程。

1.1.1 决策机制

开发人员通过修改框架的各项参数定义时间约束条件。与强制执行任务不同，可选执行任务的时间约束条件不是基于最差情况，而是根据任务平均执行时间设定的。这使得更复杂、更耗时的可选计算可被考虑作为候选执行任务。

决策机制先进行预评估，测试框架的 CPU 分配情况，并测定 CPU 分配是否在某个值以下。如果否，则 CPU 无法提供所需的执行时间，不必进一步评估。如满足该值，则框架计算可用的剩余时间并评估哪些现有的可选执行任务能被运行。如任务需要的最大运行时间足够，则该任务被运行。然后从全部可用时间中减去这个最大时间并评估下一个可选任务。综上，即可建立所有可执行的可选任务列表。

1.1.2 运行机制

将系统的可选执行任务定义为方面，使用面向方面技术提供的动态机制，在运行时对这些方面编织和解编织。一个方面由一个切入点和一个通知组成。切入点定义了程序中的执行点，通常是方法的调用，而通知则定义了要执行的代码。所有的可选方面则共享它们在框架中的横切点，但按照自己的方式执行通知。动态方面编织使得框架的运行机制得以实现。由决策过程决定哪些可选方面被展开，然后横切点所定义其上的方法被调用。方面执行完毕后，方面将被解除编织。

**1.2 框架实现**

框架建立在 AspectWerkz 上，它是提供动态编织功能的面向方面工具。系统运行时，框架通过读取使用者提供的声明文件，加载各项约束条件，然后利用决策模块选择和检索各个可选方面，再由运行模块进行方面编织。框架结构如图 2 所述，该框架对开发人员屏蔽了面向方面程序设计的细节。因为，开发人员仅需使用 Java 语言编写强制执行和可选执行的任务代码并定义约束条件。框架将可选执行任务转换为方面，并在运行时进行编织。

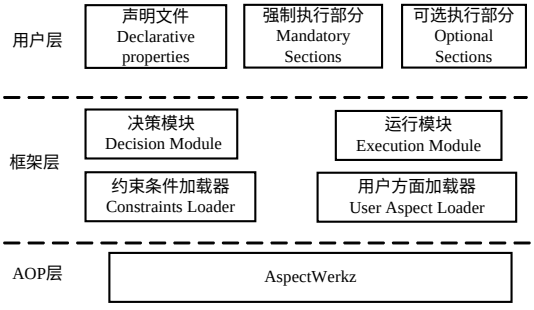


**图 2 框架的结构**

框架结构用 UML 模式图描述。为增强可读性，将模式图分为图 3 和图 4 两部分。图 3 描述实时系统的参数与可选方面的实现，其参数通过图 2 中约束条件加载器来实现；可选方面通过图 2 中的用户方面加载器来实现。图 4 描述了编织、解编织及决策的实现，其编织和解编织通过图 2 中的执行模块和决策模块来实现；决策机制通过图 2 中的决策模块来实现。

如图 3 所示，接口 InterfaceRealTimeParameters 指定了获取各时间参数的方法，执行这些方法可以检索时间约束信息。用户可通过操作声明文件很容易地扩展、修改和移除这些约束条件。

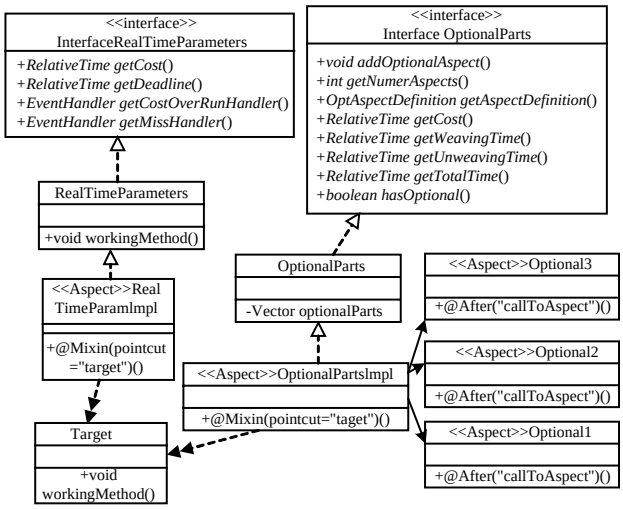


**图 3 实时系统参数与可选方面**

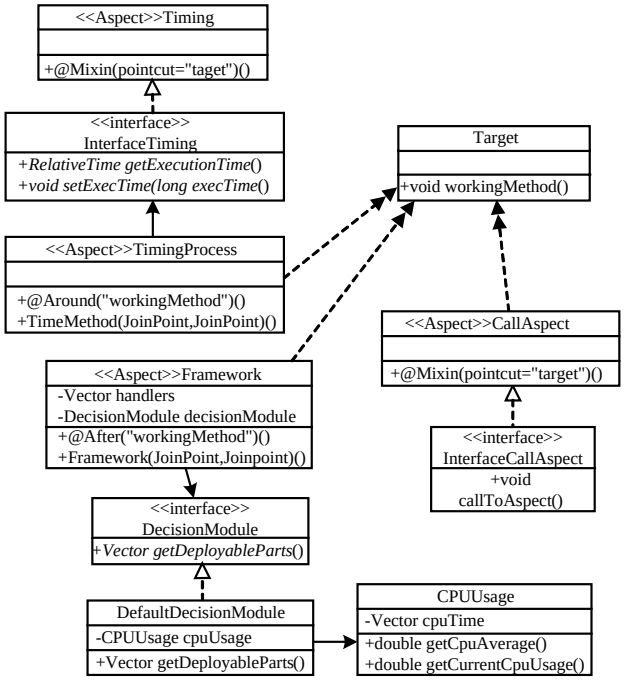


**图 4 编织/解编织模块与决策模块**

采用同样的方法，可选方面能被引入到目标任务中。每个目标任务必须指定框架定义的接口 InterfaceOptionalParts。为便于开发者工作，框架中实现了这些方法并创建了一个抽象类的对象。

开发者仅需继承 OptionalParts 类并通过 addOptional Aspect()添加自定义方面，AspectWerkz 底层的方面加载机制对使用者透明。开发人员不需理解 AOP，仅需用 Java 编码指定强制执行部分和可选执行部分，以及系统约束条件。由框架将可选任务转换为方面并将其动态织入目标方法中。

如图 4 所示，使用者可通过框架提供的接口 Decision Module 自定义可选任务决策模块。使用 DefaultDecision Module 调用类 CPUUsage 检索 CPU 平均利用率。由内部线程 CPUUsage.GetCpuUsage 更新该类的返回值，该返回值被用来做预判断。如果可分配的 CPU 大小足够，DefaultDecisionModule 将选择可行的可选任务，并调用上文提到的所有接口程序：调用 InterfaceRealTimeParameters 以得

到分配给任务的时间；调用 InterfaceTiming 检索执行时间；调用 InterfaceOptionalParts 以决定哪些可选方面将被执行。方面 Framework 在每个强制任务后运行。该方面调用 DecisionModule，检索可执行方面列表，对各可选方面进行编织。然后调用 callToAspect()方法，触发各方面并最终在各方面执行完后解编织。

## 2 测试

为确定 AspectWerkz 在实时系统中对方面进行编织和解编织的时间需求，本文设计了如下测试以评估方面部署所需时间及确定哪些参数对部署时间的影响比较大。

该测试在一个带有 Java 虚拟机的通用操作系统上进行。使用相同的计算机，并且运行同等数目的程序，以保证测试在可比较的环境下进行。

### 2.1 测试参数

测试目的是确定动态编织哪些参数影响编织与解编织时间。该参数包括：(1)通知的数目：比较编织 1 个仅包含单个通知的方面和 1 个包含 20 个通知的方面所需时间。(2)目标方法的数目：单独的 1 个方法或多目标方法(5 个)。(3)通知的类型：联接点之后，联接点周围和联接点之前运行通知。(4)联接点类型：方法，控制器和字段。(5)规定切入点所使用的表达式类型：通配符。这项测试的目标是明确使用一个模糊的规定表达式是否比使用一个精确的规定表达式需要更多时间。(6)通知或方面的规模大小：一个空方面，一个大规模的方面及一个有大规模通知的方面。(7)通知的参数：无参，1 个静态参数或者 1 个动态参数。

### 2.2 测试结果

每个测试情况被写入 1 个单独的方面且独立执行。本文设计了一个专门的方面用于计时，对每个参数进行 100 次同类测试并返回 1 个平均值结果。测试机器是 1.07GHz Power 型个人计算机，使用 10.4.1 版 Mac 操作系统，结果单位是 ms。如表 1～表 7 所示，编织比解编织耗时多一些。然而，差别不明显，二者都不可忽略。

在联接点之前、之后或周围执行通知，其所需编织时间几乎相同，差别不超过 4%；对于不同类型的联接点，无论是一个构造器、字段，还是方法，对其编织/解编织用的时间上几无差别；在定义切入点规约使用的表达式类型上，无论精确表达式(用完整的目标方法名表示，如 core.Target.foo())，或不精确表达式(用通配符表示，如*.foo())，在编织/解编织用时上差别也很小；编织/解编织带动态参数的通知与带静态参数的通知相比，时间几乎一样；代码的规模大小对于编织/解编织的用时，影响也有限。在测试中，本文增加方面的规模，从 1 行代码到 32 600 行代码，编织时间相应地从 77.3ms 增加为 100.5ms，代码规模增加了 32 600 倍但用时仅增长 30%。

在变化通知和目标方法数量时，即比较 1 个通知与 20 个通知时，或比较 1 个方法与 5 个方法的情况时，编织/解编织用时均相差很大。

**表 1 通知类型的测试数据**

| 通知类型 | 编织/ms | 解编织/ms |
|---|---|---|
| 联接点之后运行 | 76.9 | 77.0 |
| 联接点之前运行 | 79.9 | 80.0 |
| 联接点周围运行 | 79.4 | 79.5 |

**表 2 联接点类型的测试数据**

| 联接点 | 编织/ms | 解编织/ms |
|---|---|---|
| 方法 | 86.5 | 86.6 |
| 字段 | 86.2 | 86.3 |
| 控制器 | 87.1 | 87.2 |

**表 3 切入点的测试数据**

| 切入点 | 编织/ms | 解编织/ms |
|---|---|---|
| 模糊定义 | 27.6 | 27.6 |
| 精确定义 | 27.3 | 27.3 |

**表 4 通知参数的测试数据**

| 通知的参数 | 编织/ms | 解编织/ms |
|---|---|---|
| 无参 | 77.2 | 77.2 |
| 静态参数 | 80.5 | 80.6 |
| 动态参数 | 77.9 | 78.0 |

**表 5 方面的规模测试数据**

| 通知的规模 | 编织/ms | 解编织/ms |
|---|---|---|
| 1 行代码 | 77.3 | 77.4 |
| 32600 行代码 | 100.6 | 100.7 |

**表 6 通知数目的测试数据**

| 通知的数目 | 编织/ms | 解编织/ms |
|---|---|---|
| 多通知(20 个) | 155.1 | 155.2 |
| 单个通知 | 77.6 | 77.6 |

**表 7 目标方法数的测试数据**

| 目标方法数 | 编织/ms | 解编织/ms |
|---|---|---|
| 多目标方法(5 个) | 271.8 | 271.9 |
| 单目标方法 | 79.4 | 79.5 |

其他参数的影响则很小。通过控制这两个变量，可在方面编织时间上得到期望的可预测性，这说明面向方面技术可应用于非精确计算。由于测试环境采用通用操作系统，任何环境变化，如改变程序或运行于这台计算机上的某服务等，均会影响标准时间，因此本结果是相对的。

## 3 结束语

测试结果表明，该框架通过应用非精确计算和面向方面技术增强了实时系统的柔性和动态适应性，具有应对可预测变化需求的优势，但仍需完善：(1)拟用直观和简洁 XML 来定义可选执行部分的声明，通过 XML 文件保存强制部分以及相关的可选部分的声明信息。(2)使框架可按需要在运行时重装声明文件，动态地更新信息和约束条件，当系统要改变规约时不必重启动。(3)应继续改进判定算法，以能够基于系统环境信息得到更精确和优化的决策。

## 参考文献


1 Shih W K, Liu J. On-line Scheduling of Imprecise Computations to Minimize Error[C]//Proceedings of the 13th IEEE Real-tme Systems Symposium. 1992: 280-289.

2 Aydin H, Melhem R, Mosse D. Optimal Scheduling of Imprecise Computation Tasks in the Presence of Multiple Faults[C]//Proc. of the 7th IEEE International Conference on Real-time Computing Systems and Applications. 2000: 289-296.

3 张尧学, 方存好, 王 勇. 非精确计算中基于反馈的 CPU 在线调度算法[J]. 软件学报, 2004, 15(4): 616-623.

4 王 亮, 雷 航, 桑 楠. 非精确任务集的容错 EDF 调度[J]. 计算机工程, 2004, 30(23): 56-152.

5 Suvee D, Vanderperren W, Jasco V J: An Aspect-oriented Approach Tailored for Component Based Software Development[C]// Proceedings of the 2nd International Conference on Aspect-oriented Software Development. New York: ACM press. 2003: 21-29.

6 Bonér J, Vasseur A. Aspectwerkz-Plain Java AOP[EB/OL]. (2005-06). http://aspectwerkz.codehaus.org/.