基于模型检查实现 J2EE 规范的实例研究*)

李 彦[1,2,3] 张文博[1,2,3] 陈宁江[1,2,3]

(中国科学院软件研究所软件工程技术中心 北京 100080)[1]

(中国科学院软件研究所计算机科学重点实验室 北京 100080)[2] (中国科学院研究生院 北京 100080)[3]

摘 要 J2EE规范描述了当前开发应用服务器和分布式多层应用所遵循的技术蓝本。然而,它所使用的自然或半自然语言描述方式并不严格,易产生二义性,会影响J2EE应用服务器实现的正确性和应用服务器之间的兼容性。针对这一问题,本文以EJB2.1规范中的Timer Service为例,研究了一种基于模型检查技术设计与实现规范方法。首先根据规范的描述提出Timer Service的形式化模型,定义了Timer Service的行为;然后使用模型检查工具SPIN对模型进行分析与验证,不仅证明了模型符合规范要求,而且发现并修正了规范中不严格的描述带来的缺陷。以该模型为基础导出了Timer Service的一种设计方案,这种设计已经在中科院软件所研制的OnceAS应用服务器中得到实现,并在J2EE1.4兼容性测试中证明了其正确性。

关键词 J2EE规范,模型检查,SPIN

A Case Study in Implementing J2EE Specification Based on Model Checking

LI Yan[1,2,3] ZHANG Wen-Bo[1,2,3] CHEN Ning-Jiang[1,2,3]

(Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100084)[1]

(Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080)[2]

(Graduate School of the Chinese Academy of Sciences, Beijing 100039)[3]

Abstract J2EE specifications provide the technique blue-prints of the current development of application servers and distributed multi-layer applications. However, the J2EE specifications are expressed in nature/half-nature language, which brings vulnerabilities such as ambiguity, incorrectness and incompatibility. In this paper, a specification design&implementatation approach based on model checking technology is discussed to solve such problems. Semantic models are applied on the Timer Service in EJB™ Specification 2.1 and formally verified. Using the SPIN model checker, defects both in the design of the model and in the expressions of J2EE specification are discovered and corrected. The process resulted in a faultless and highly compatible model, and based on which, a design of Timer Service is exported. This design has implemented in OnceAS and passed the J2EE1.4 CTS test.

Keywords *J2EE specification, Model checking, SPIN*

1 引言

中间件在多层计算模型中处于核心的地位。其中,J2EE[1]技术以其开放性和跨平台性成为企业级中间件平台的首选方案,它通过一系列的技术规范定义了统一的组件模型和完整的开发框架。但作为J2EE技术的载体,J2EE规范使用自然语言和半自然语言进行表述。这样的表述方式虽然简单易懂,但也存在明显的缺陷:1)非严格的描述方式很容易使规范内容产生二义性。2)规范使用方法序列和示例代码来描述各组件之间的交互,而不是使用严格定义的规则,这使得规范很难完整描述其问题域中的情况。因此,如果直接以J2EE规范为蓝本进行中间件设计,很难完全保证结果的兼容性和正确性。

上述问题已引起了研究者的关注。文[2~4]使用模型检查[5]的方法对规范进行了验证,并找出了规范的疏漏,但他们的工作并不涉及后续的中间件设计与实现;文[6~8]将形式化分析用于产品实现后的功能测试,而非设计前的指导。本文将关注一种基于模型行为分析的实现J2EE规范的方法。我们首先根据规范中的描述对问题域进行建模,然后对这个模型进行验证。以验证后的模型为指导完成的J2EE中间件的设计不仅是完全符合规范的,同时还弥补了规范描述中存在的缺陷,大大提高了中间件的设计的正确性。

本文以EJB2.1[9]规范中的Timer Service为例结合模型检查工具SPIN[10]实践了上述的方法。首先我们使用形式化描述语言Promela[10]对Timer Service规范进行了严格的描述,提出了一个形式化的模型。然后我们使用线性时序逻辑(LTL)[11]描述了Timer Service的各种行为属性,并使用SPIN验证了这些属性。通过修改验证过程中出现的错误,我们得到了一个符合规范并且正确无误的Timer Service原型。最后,以这个模型为基础,我们快捷地导出了Timer Service的一种设计方案。

文章其后的章节组织如下:第2节简单介绍SPIN和Promela;第3节介绍Timer Service规范;第4节针对规范中存在的问题提出我们的解决方案;第5节阐述Timer Service形式化模型的设计与实现;第6节使用SPIN对模型的行为进行了验证;第7节我们以模型为基础,快速地导出了Timer Service一种高可信的设计实现;第8节结合相关的研究进行了讨论,最后总结了我们的工作。

*)973计划:网构软件中间件平台模型和框架研究(2002CB312005);863计划:网络环境的系统软件核心技术及运行平台(2001AA113010);863计划:面向新型ERP的可重配Web应用服务器的研究与应用(2003AA413010)。李 彦 硕士研究生,主要研究领域为分布对象计算;张文博 博士研究生,主要研究领域为分布对象计算;陈宁江 讲师,主要研究领域为分布对象计算。

2 SPIN/Promela

模型检查是软件工程领域的一项重要技术，近年来出现了许多有影响力的模型检查工具，其中 SPIN 因其能够自动执行并给出反例而成为其中的佼佼者。

SPIN 针对有限状态分布式系统。它使用 Promela 语言对并发系统进行形式化描述，并将系统需要满足的属性转化成线性时序逻辑(LTL)或者是 assert 断言，然后通过穷举系统的所有状态来确认系统是否满足这些关键属性，如果该属性得不到满足，SPIN 将会给出错误情况的执行轨迹。

Promela 是 SPIN 使用的建模语言。它是一种类 C 的形式化描述语言，提供消息信道、接受/发送原语以及并发进程的同步互斥等诸多支持，以描述一个并发的分布式系统。Promela 模型将系统划分成一系列的活动对象(proctype)，活动对象之间通过消息信道(channel)进行通信，Proctype 是一个独立运行的实体，它们之间通过消息传递来进行通信，流程之间按照完全随机的次序运行。关于 Promela 的更详细的语法规则请参见文[10,11]。

3 Timer Service 规范

Timer Service 是 EJB2.1 规范中新增的内容，它是由 EJB 容器管理的具有事务性的定时提醒服务。EJB 提供商可以将自己的 EJB 注册到一个或多个定时器 Timer 上，使定时器在一个预设的时间间隔之后或者一个预设的时间点回调 EJB 中的特定方法。

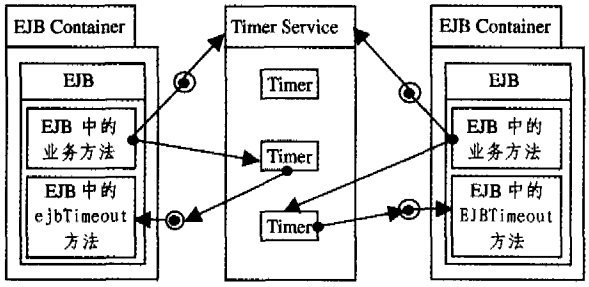

图 1 Timer Service 的结构示意图

图 1 展现了 Timer Service 的体系结构和主要参与者。Timer Service 是一个独立的服务，它管理着若干个 Timer。每一个活动的 Timer 都对应着唯一的一个 EJB。EJB 运行于 EJB container 之中，通过 EJB Container 获得 Timer Service 的引用。然后调用 Timer Service 的创建 Timer 的方法创建一个属于自己的 Timer。创建时，EJB 需要设定 Timer 的触发时机。当触发时机满足之后，Timer Service 会提醒 Timer 回调 EJB 中的 ejbTimeout 方法，执行 EJB 提供商预设的操作。EJB 还可以通过撤销这个 Timer。Timer 被撤销后，ejbTimeout 方法将不会被调用。

我们通过 Timer 状态变迁图和 Timer Service 的关键操作的顺序图来展现 Timer Service 的行为特征。

由图 2 和图 3 可以看出，Timer 的状态变迁可以是由多种因素促成的：可能是一个方法调用或返回，也可能是一个事务的提交或回滚。Timer 的状态变迁序列实际上就是 Timer Service 相关的一系列动作的时序组合。Timer 的生命周期起始于 EJB 对 Timer 的创建，并终结于两个可能的状态：Canceled 和 Expired。Timer 的撤销是由 EJB 发起的，EJB 调用 Timer 的 cancel()方法，如果撤销所在的事务成功提交，Timer 将进入 Canceled 状态。过期是 Timer 自身主导的状态变迁。“只触发一次”的 Timer 在超时提醒所在的事务提交后或者重试一次超时提醒之后，将会进入 Expired 状态。Timer 的创建、撤销和超时提醒都是在一个事务中进行的。事务的提交或回滚会带来不同的状态变迁。但在图 2 中，无论“Retry Timeout”所在事务提交或者回滚，都会进入同一个状态——“Finish Timeout”。这是因为规范要求如果 Timer 提醒 EJB 的操作所在的事务回滚，则 Timer 至少重试一次。在这里，我们规定 Timer 重试一次后无论事务是否提交，均不再重试。

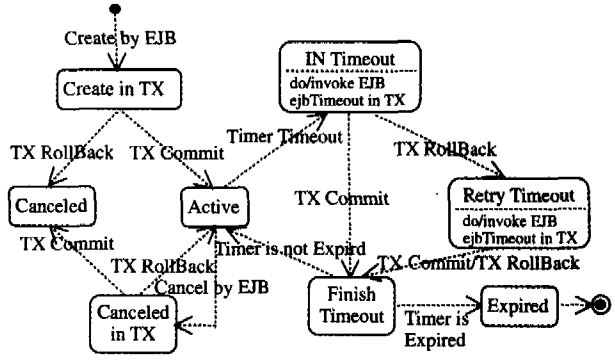

图 2 Timer 的状态变迁图

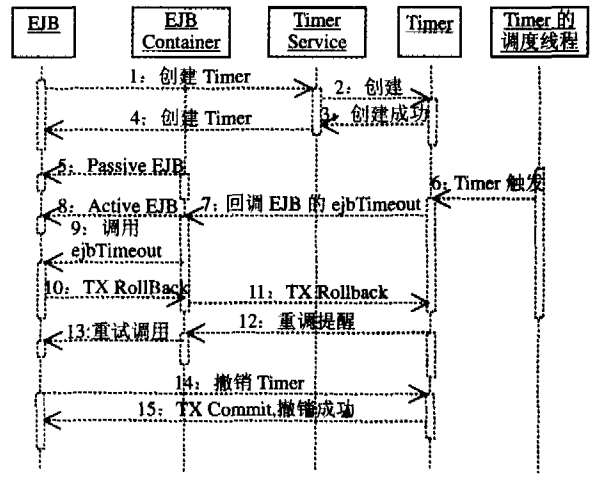

图 3 Timer Service 的操作顺序图

4 问题分析及解决方法概述

J2EE 规范中存在的问题主要集中在规范描述的正确性，无二义性以及完备性，上述问题在 Timer Service 规范中十分典型：首先 Timer Service 规范的要求中存在着很强的时序约束，而规范中却存在不少的模糊表述；其次 Timer 的触发和撤销操作是由两个不同线程中的对象发起的，而 Timer 对于 EJB 的回调过程中也交织着 EJB 容器对于 EJB 的生命周期管理，在 Timer Service 的规范中，对于这些并发中的事件序列只是做了简单的说明，并没有做出严格的定义。基于上述原因，如果仅仅以规范为指导，极可能由于不恰当的设计中引起并发的线程对于 Timer 这一公共资源的错误争用，最终导致死锁或者不正确的操作结果。为了解决这些问题，我们采用如下方法：

(1)将自然语言表述的 J2EE 规范转化为形式化模型。由于基于无二义性的形式化语言进行描述，因而该模型消除了自然语言中可能存在的二义性的情况。

(2)针对这个模型使用模型检查工具验证该模型在其状态空间中其是否满足规范规定的各种属性，例如有无死锁和规范中的各种时序性质。

(3)如果验证中发现模型违犯了某条性质,则根据检查工具提供的错误日志修改模型中的问题,提升模型正确性和完备性。

(4)当模型通过验证修改了其中存在的缺陷之后,该模型就可以被认为是一个具有较高正确性和完备性,并且无二义性的系统描述。以这个模型为基础,我们可以导出系统的UML设计。由于该设计源自经过验证的模型,因此它也避免了模型检查过程中发现的各种错误。

5 Timer Service的形式化模型

5.1 模型的设计

我们使用Promela语言描述Timer Service的模型。图4展现这个模型的架构。

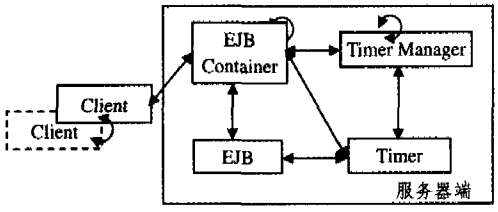

图4 Timer Service的Promela活动对象模型

该模型共由5个独立的活动对象构成,每个对象都对应着相应的Promela流程。EJB Container对象模拟EJB容器,它拦截并转发EJB相关的方法调用,并向EJB提供运行时的服务。Timer Manager对象负责Timer的创建、调度和删除。EJB对象则接受Client的请求,发起对Timer Manager和Timer的操作。Timer对象触发EJB的ejbTimeout方法,同时受Timer Service的调度,也可以被EJB撤销。作为服务器端的流程,上述对象既不能预知客户端请求的内容,也不能影响请求的顺序,而只能被动响应客户端请求,它们都不应成为测试流程的最初发起者。所以,模型引入了一个Client对象,专门负责发起各种请求,这样使得对于服务器端组件的验证更加准确。在模型初始状态中,Client, EJB Container和Timer Manager应该已处于活动状态,所以它们被标识为模型的初始流程。

5.2 模型的实现

在Timer Service的Promela模型中,活动对象的实现方式依据它们的行为模式可以分为两类:第一类的生命周期是一个循环执行过程:在处理完一个请求之后,其流程回到初始状态,等待下一个请求,EJB Container, TimerManager和EJB属于种类型;另一类则包括Client和Timer,它们的执行序列是一个线性的过程:流程从初始状态启动后,将不会再回到初始状态。

5.2.1 循环流程的活动对象实现

对于这一类对象,我们使用do……od结构来实现它们的主体框架。do……od中的每一个分支都对应着一个可以被调用的方法,分支的执行条件就是信道中出现了该方法对应的信息,分支的实体就是方法的具体操作。当方法调用者向信道内发出调用方法对应的信息时,方法执行者接受该信息,并执行相应的操作。在模型的实现中,我们消除了方法执行过程中各种不正确的同步等待,避免了由信道同步带来的死锁。下面以EJB流程为例,说明了do……od结构的应用:

```
proctype EJB(){
do
:: mthd_EJB_Container?? CREATETIMER-> ……/* 创建 Timer */
:: mthd_EJB_Container?? RETURN_TIMERSERVICE-> ……/* 从 Container 获得 Timer Service 的作用 */
……
od}
```

规范要求Timer的核心流程在事务中执行,所以事务的提交或回滚都直接影响模型的状态变迁。我们在模型中使用if……fi结构模拟了这一点。以ejbTimeout方法的调用为例,我们在EJB中该方法处理流程的最后加入了如下代码。这些代码将向一个专有的信道中随机地决定发送一个事务提交消息或者一个事务回滚消息。Timer发起调用后将监听这一信道,并根据收到的消息决定自己的状态变迁。

```
if
::TRANSACTION! EJBTIMEOUT_TXCOMMIT;
::TRANSACTION! EJBTIMEOUT_TXROLLBACK;
fi;
```

我们还使用Promela中的else关键字定义各种方法处理之间的级别。例如,在TimerManager流程中,为了模拟Timer的超时提醒,我们设定当Timer处于Active状态时,TimerManagr就可以触发Timer。但是我们在验证过程中发现这样有可能引起Timer的循环触发,而其它流程根本得不到执行的机会,最终导致活锁。为了杜绝这种情况的出现,我们使用else关键字对流程中的分支进行优先级分级:当信道中不存在其它方法的调用时TimerManager才会触发一次Timer,否则将优先执行其他方法。这样既避免了活锁(live-lock),也给Timer的触发带来一定的间隔,增强了模型的真实度。

```
active proctype Timer Manager(){
do
 ::if
  ……          /* Timer的创建等高优先级方法 */
  fi
  ::else->…… /* Timer的超时提醒,低优先级方法 */
od}
```

5.2.2 线形流程的活动对象实现

我们采用线性结构实现第二种对象的流程。Timer流程具有复杂的状态变化,包括众多的分支选择和跳转,不同信道在不同状态下的输入,会使Timer发生不同的状态变化。我们按照第3节中给出的行为规约,使用goto关键字实现了Timer的生命周期中各种状态的转化。

```
proctype Timer(){
……
timer_actives; mthd_TimerService_Timer? EJBTIMEOUT-> goto in_timeout;  /* 提醒 EJB */
in_timeout;mthd_Timer_Container! EJBTIMEOUT;          /* 在事务中提醒 EJB */
if   /* 根据事物是否提交决定 Timer 的状态变迁 */
  ::TRANSACTION?? EJBTIMEOUT_TXCOMMIT->gotl finish_timeout   /* 事务提交 */
  :: TRANSACTION?? EJBTIEOUT_TXROLLBACK-> goto retry_timeout   /* 事务回滚 */
fi;
……}
```

Client流程需要顺序的发起请求、接受结果,但作为整个测试的最初发起者,它决定着模型的行为。为了全面详细地分析验证Timer Service的模型,我们提出了三种不同的Client流程。1)Client-1,一个标准的客户端,它顺序执行Timer的创建,撤销,EJB的删除;2)Client-2,它创建Timer之后,并不撤销Timer,而是直接删除EJB;3)Client-3,它创建Timer之后,既不撤销Timer,也不删除EJB,而只是让Timer在服务器端循环地进行超时触发。对于这些Client流程有两点需要说明:1)从Client角度,Timer的创建和撤销似乎是两个连续进行的操作,没有给Timer留出触发的时间。但是在SPIN模型验证的过程中采取的是穷举方式,因此在这两个操作之间,SPIN会插入所有可能的操作来验证模型在各种状态下都是无死锁的,其中自然包括Timer的超时提醒;2)EJB规范的Timer Service规范中对会话Bean、实体Bean和消息驱动

Beans 设置的处理流程基本相同，唯一的特殊之处是规范要求实体 Bean 实例在删除时也删除对应的 Timer。因此，我们的 EJB 流程也采用一个通用的模型，并不指代哪种具体的 EJB，只是在 ejbRemove 操作中实现了规范对于实体 Bean 的要求，删除了所对应的 Timer。

6 Timer Service 模型的行为分析

首先进行模型的死锁检验：我们分别使用上一节中提到的三种不同的客户端在 SPIN 中对模型的状态进行了遍历(spin-a 命令)，执行结果证明了模型是无死锁的。

规范中对一些特定事件的发生顺序做出了严格的规定。我们从这些描述中抽象出 LTL 表述的形式化命题，再使用 SPIN 将 LTL 转化成 Promela 代码并加入模型中进行验证，以证明模型是否满足规范要求的行为属性。

例如，当验证“EJB Contianer 在 ejbPassivate 和 ejbTimeout 之间至少调用一次 ejbActivate”这一性质时，我们首先从命题描述中抽取所有涉及到的时序状态(动作)：

```
# define p3 = mthd_EJB_Container?? [ EJBTIMEOUT] / * EJB Container 调用 EJB 的 ejbTimeout 方法 * /
# define p4 = EJB_State == PASSIVED  / * EJB 的状态为 PASSIVED * /
# define p5 = EJB_State == ACTIVED  / * EJB 的状态为 ACTIVED * /
```

然后使用 LTL 将上面的定义组合成一个时序描述。

属性 1 $[](p^4 \&\& p^3) \rightarrow (! \ p^3 U p^5)$(F1)

对于上面的 LTL 命题，我们使用 spin - f 命令将其转化成 Promela 语言中的 never claim(实际上就是一个 Büchi 自动机)，然后将 never claim 与模型放在同一个文件中，使用 spin - a 遍历它们的组合状态，证明了模型是满足该属性的。

spin-f“$! [](p^4 \&\& p^3) \rightarrow (! \ p^3 U p^5))$”

注意到命题在转化时被置为否，这是由于 never 断言只在命题被满足的时候才会抛出异常，为了验证以[](always)标示的永真命题的正确性，我们采用了反证法：验证它的否命题永远为假，即命题的 never 断言是否永远不报错。

使用同样的方法，我们还验证了 Timer Service 另外一些行为属性也是可以被满足的。这些属性包括：

属性 2 如果 Timer 的创建所在的事务回滚，Timer 的创建也将无效(Timer 进入“CANCELED”状态)。

$[](Timer\_Creation\_TxRollback) \rightarrow (\langle\rangle Timer\_Canceled)$(F2)

属性 3 如果撤销 Timer 的事物回滚。Timer 的撤销也将无效(Timer 进入“ACTIVED”状态)。

$[](Timer\_Cancel\_TxRollback) \rightarrow (\langle\rangle Timer\_Actived)$(F3)

属性 4 如果 ejbTimeout 方法所在的事务回滚，Timer 至少重试调用一次 ejbTimeout(Timer 进入“RETRY_TIMEOUT”状态)。

$[](ejbTimeout\_TxRollback) \rightarrow (\langle\rangle Timer\_Retry\ Timeout)$(F4)

属性 5 当实体 Bean 被删除的时候，它所对应的 Timer 也会被删除。

$[](EJB\_Remove) \rightarrow (\langle\rangle Timer\_Removed)$(F5)

属性 6 EJB 在创建 Timer 之前，必须从 Contianer 获得 Timer Service 的引用，并使用 Timer Service 创建 Timer。

$[]((EJB\_CreateTimer) \rightarrow ((! \ EJB\_GetTimerService) \ U \ EJB\_CreatTimer)$(F6)

$[]((EJB\_CreateTimer \ \&\& \ \&TimerCreated) \rightarrow ((! \ TimerService\_CreateTimer) \cup TimerCreated)$(F7)

然而，并不是所有的性质都得到了满足，一些基本的关键行为属性在这个模型上却并不成立。例如“如果 Timer 超时，ejbTimeout 方法就会被调用。”

属性 7 $[](TimerManager\_TimerTimeout \rightarrow EJB\_ejbTimeout)$(F8)

验证过程中，我们使用 SPIN 的 weak-fairness flag 屏蔽了活锁可能带来的影响。但是，这条属性仍然被违反了。最后我们通过查看 SPIN 的验证记录，发现当 Timer 发生超时和 Timer 发出 ejbTimeout 调用这两个操作的空隙中，EJB 发出撤销 Timer 的调用得到了执行，使 Timer 对 EJB 的超时提醒无从发出，导致了上述情况。

同样，另外一条基本属性也没有得到满足：

属性 8 $[](Timer\_Canceled \rightarrow ! \ EJB\_EJBTimeout)$(F9)

即便 Timer 被撤销之后，EJB 仍然可能收到 ejbTimeout 的调用。究其原因，EJB 发出撤销 Timer 的指令和 Timer 被真正撤销之间存在一定时间空隙，如果 Timer 在这一空隙中发生了超时并发出了 ejbTimeout 调用，改调用将会得到执行。虽然 Timer 随后进入了 canceled 状态，EJB 仍然收到了一次超时提醒。

出现上面的情况，究其原因是 EJB 撤销 Timer 动作与 Timer 超时提醒动作发生时间上的重叠。由于 Timer 的撤销和超时提醒是由两个完全独立的线程(EJB 和 TimerManager)控制的，如果不加控制的任由两个线程对 Timer 就进行无序的并发争用，必然会产生不确定执行结果——Timer 有可能仍然发起一次 ejbTimeout 调用，也有可能被先行撤销而来不及发起超时调用。然而，这些情况在规范中并没有得到明确的说明，很容易在规范的实现和 EJB 的设计过程中被忽略，从而造成不可预知的后果。

为了避免这种情况的发生，我们在模型中设置了一个互斥信号量 mutex，并修改了 Timer 和 TimerManager 流程，以同步 Timer 的撤销和超时提醒，并限制 Timer 在收到撤销指令之后不得发出新的超时调用。

为了证明信号量的设计是有效的，我们提出了两条属性来对其进行检验：

属性 9 在 Timer 超时至其完成超时提醒之间的时间 EJB 不得发起的撤销 Timer 的操作。

$[](TimerManager\_TimerTimeout \rightarrow (! \ EJB\_Cancel\_Timer \cup Time\_TimeoutFinishes))$(F10)

属性 10 EJB 发起撤销 Timer 的操作之后，TimerManager 不得提醒该 Timer 进入超时状态。

$[](EJB\_Cancel\_Timerc \rightarrow (! \ TimerManager\_TimerTimeout \cup Time\_CancelFinished))$(F11)

```
active procype TimerManager(){
do
    ……
    if::(mutext==0)->mutext=1;……/ * 触发 Timer * /
    ::(mutext==1)->……/ * 继续等待 * /
fi
od}
proctype Timer(){
……
actived;mthd_EJB_Timer? CANCELTIMER->if::(mutext==0)->
mutext=1;……/ * 撤销 Timer * /goto canceldd;
```

```
::(mutext==1)->gotl actived /*继续等待*/fi
.....
canceled;mutext=0;      /*无论 Timer 的撤销操作是否成功,设置标志位*/
finish_timeout;mugtext=0;  /*无论 Timer 的超时触发是否成功,设置标志位*/
}
```

通过 SPIN 的验证,修改过的系统满足属性 9 和属性 10,证明该信号量可以有效的避免它们的无序并发。而且我们的模型还通过了属性 7 和属性 8 的验证,这证明修改也弥补了规范描述中的漏洞。

7 Timer Service 的设计导出

通过前面的分析,我们获得了 Timer Service 的一个具有较高正确性和完备性,并且无二义性的形式化描述,它不仅准确描述了 Timer Service 各部分的功能,而且还正确的定义了它们之间的交互。同时,模型还在实验规范要求的基础上避免了规范描述中的缺陷。下面,我们就以这个模型为基础导出 Timer Service 的一种设计。首先,我们给出从 Promela 模型中导出 UML 类图的算法的描述:

```
UML_Class_Diagram exportClassDiagram(Promela_Model model){
    While(Not all the Proctypes, Channels and Messages in model are marked){
        Scan the un-marked part of mldel;
        If(there is a Proctype that is mot marked in model){
            Add a corresponding class in UML_Class_Diagram;
            Mark this Proctype to be"already checked"in model;
        }
        If(there is a Channel that is not marked in model){
            Check who ar the two proctypes that this channel connects;
            Add relationship between the corresponding classes in UML_Class_Diagram;
            Mark this Channel to be"already checked"in model;
        }
        If(there is a Message that is not marked in model){
            Check which procedure receives this message;
            Add a method in the corresponding cladd in UML_Class_Diagram;
            Mark this Message to be"already checked"in model;
        }
    }
    Return UML_Class_Diagram;
}
```

根据上述算法,Promela 模型中的活动对象被映射为类图中的类或接口,活动对象之间的消息信道被归结为类之间的关联,而消息信道中传递的消息则被视为类中暴露出的公共方法。图 5 展现了我们从 Promela 模型中导出的 Timer Service 类图。

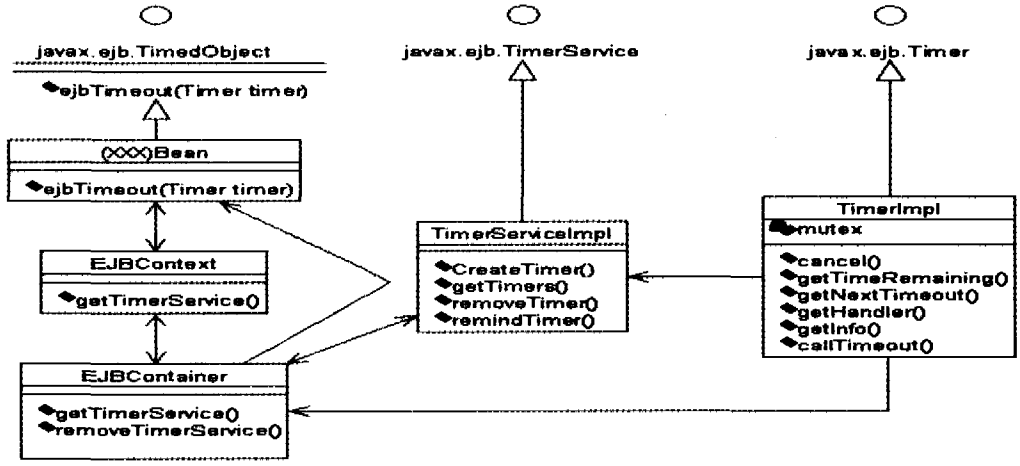

图 5 Timer Service 的核心类图

在确定了 Timer Service 的静态结构之后,我们继续给出 Timer Service 各个类之间的交互。可以看出,具体设计的静态结构实际上直接脱胎于前文中的原型,我们只是将其中抽象的描述具体化和实例化。设计中的每个类都可以在模型中找到相应的 Proctype,类的方法也是完全按照模型中流程之间传递的消息定义的,所以设计和原型在行为上是完全等价的,Timer Service 核心行为可以从原型中流程之间的交互过程直接导出。下面给出导出算法的描述。

```
UML_Sequence_Diagram exportSequenceDiagram (PROMELA_Model model){
    While(Not all the message send/receice action are marked in model){
        Scan the un-marked part of model;
        If (there is a message send action fron Proctype A to Proctype B that is not marked in model){
            Add the corresponding method invocation from cladd A to cladd B in UML_Sequence_Diagram;
            Mark this message send action to be"already checked"in model;
        }
        If (there is a message receive action from Proctype A to Proctype B is not marked in model){
            Add the following operations of this message receive action in model into the thread of cladd B in
            UML_Sequence_Diagram after the corresponding method invocation;
            Mark this message receive action to be "already checked"in model;
        }
    }
    Return UML_Sequnce_Diagram;
}
```

图 6 是根据上述算法给出的 Timer 撤销和超时触发过程顺序图。但与第 3 节中的顺序图不同,图 6 基于一个经过严格验证的形式化模型,而不是一个自然语言描述的规范。原来的顺序图中存在的问题已经根据模型验证及设计导出时进行了修正。

这里我们给出的是 Timer Service 核心类的结构和核心行为的定义。但作为一个完善的设计,我们还需要补充一些辅助类和一些次要的流程,例如 Timer Handler 及其相关处理。这些新增加的内容并没有影响 Timer Service 的核心流程,从而无损于设计的行为正确性。

上述 Timer Service 设计已经在中科院软件所软件工程中心开发的 Web 应用服务器 OnceAS[18] 中付诸实现,并通过了 Sun 公司的 J2EE1.4 兼容性测试(J2EE1.4 CTS)[12],该测试针对 J2EE 规范中每一个功能需求点都设计了严格的测试用例,其中针对 Timer Service 就提出了多达 110 个测试用例,涵盖了规范中提及的所有功能点。同时,该设计还通过了我们针对规范中忽略的特殊情况编写的测试用例。这些测试

用例模拟了模型验证过程中曾经出错的场景，例如我们在测试用例中特意使 Timer Service 的超时提醒和 Timer 的撤销操作并发执行，验证我们的设计是否避免了第 6 节中讨论的不确定性。

上述测试有效地证明了我们的设计不仅是符合规范的，而且还规避了规范中可能存在的缺陷，因而证明了我们设计具有较高的正确性和完备性，也很好地展现了由形式化模型导出设计这一方法的正确性和有效性。

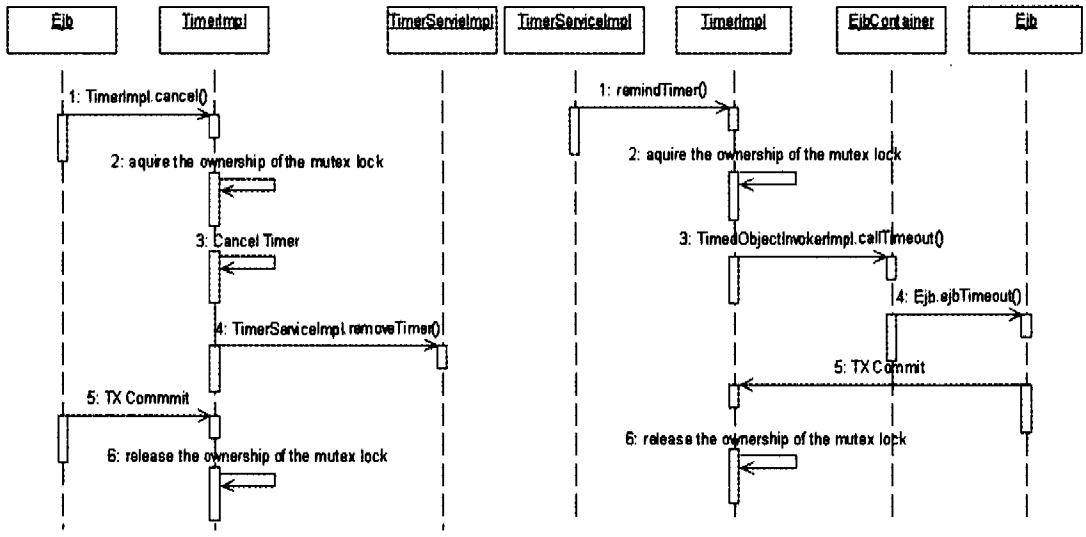

图 6　Timer 超时触发和撤销的典型流程的顺序图

8　相关工作

Nakamija 和 Tamai[2] 则使用模型检查工具 SPIN 和 Promela 语言对 EJB 组件的行为模式进行了形式化分析；Sousa 等[3] 使用 Wright 对于 EJB 框架进行了形式化描述，并使用 FDR 模型检查方法分析了 EJB 规范。他们最终都找出了规范描述中的缺陷。但是他们仍然停留在“如何正确理解规范”这一层面上。R. Kazhamiakin 等[4] 使用 SPIN 对一个以 Web Services 为核心的框架进行了形式化验证，但验证的重点还是软件需求的合理性。因此，他们的工作对设计实现的指导意义有限。

K. Havelund 等[6] 将模型验证技术应用到一个已经实现的系统中，他们使用 SPIN 和 Promela 将一个 LISP 语言实现的并行系统重新进行形式化描述，并发现了其中存在的死锁和错误。D. Garlan 等人[7] 直接使用 SMV 建模了一个 Pub/Sub 的系统并进行了模型检查。Inveradi 和 Muccini[8] 使用 UML 对软件体系结构进行了建模，再将可视化模型转化成形式化模型，并结合 SPIN 验证了该模型的功能。他们的工作更多的是一种后验性质的查缺补漏。而我们的工作则注重于使用通过检验的模型来导出一个正确的中间件设计。

综合前人的工作，我们将模型验证与行为分析用于指导软件的设计，即在设计之前使用形式化描述的规范原型来验证规范描述和我们的设计思路，发现其中的缺陷，解决可能出现的问题，再使用通过验证的原型导出最终的设计。这样的设计方法提升了软件设计的正确性和完备性，同时它也很好的证明了形式化验证技术对于软件开发的指导意义。

结束语　本文采用“提出模型->验证模型->导出设计”的方法，探讨了一种基于模型检查的 J2EE 规范设计方法，这种方法可以有效地避免规范的不完备和二义性对中间件设计带来的影响。本文以 Timer Service 为例，使用模型检查工具 SPIN 在 Promela 模型上验证了规范提出的各种行为属性，这种验证不仅可以验证模型是否符合规范，而且还可以发现规范描述中的不足之处。由于设计思路在模型的验证过程中已经得到了修正，所以我们在模型基础上快捷地导出一种正确的设计方案。本文的工作已经在 Web 应用服务器 OnceAS[18] 中付诸实现，并通过了权威的 J2EE1.4 兼容性认证。进一步的工作包括如下方面：1）进一步完善从 Promela 模型到 UML 设计的转化流程，包括更详尽的规则和更高效的算法。2）将验证的属性从功能性领域拓展到非功能性领域，例如 QoS 属性、实时约束等。

参 考 文 献

1　Sun Microsystems Inc. Java 2 Enterprise Edition, v1.4: http://java.sun.com/j2ee/1.4/index.jsp
2　Nakajima S, Tamai T. Behavioral Analysis of the Enterprise JavaBeans Component Architecture. Proceedings of the 8th SPIN Workshop, LNCS 2057, 2001
3　Sousa J, Garlan D. Formal Modeling of the Enterprise JavaBeansTM Component Integration Framework. In: Proc FM'99, 1999. 1281～1300
4　Kazhamiakin R, Pistore M, Roveri M. Formal Verification of Requirements using SPIN: A Case Study on Web Services. Proceedings of the Software Engineering and Formal Methods, Second International Conference on (SEFM'04), 2004
5　Visser W, Havelund K, Brat G, et al. Model Checking Programs. 15th IEEE International Conference on Automated Software Engineering (ASE'00), 2000
6　Havelund K, Lowry M, Penix J. Formal Analysis of a Space-Craft Controller Using SPIN. IEEE Trans on Software Engineering, 2001, 27(8): 749～765
7　Garlan D, Khersonsky S. Model Checking Publish-Subscribe Systems. In: JS Kim SPIN 2003, LNCS 2648, 2003. 166～180
8　Inverardi P, Muccini H, Pelliccione P. Automated check of architectural models consistency using SPIN. In: Feather M, Goedicke M. eds. Proc of the 16th IEEE Int'l Conf on Automated Software Engineering (ASE 2001). Los Alamitos: IEEE Computer Society Press, 2001. 346～349
9　Sun Microsystems Inc. Enterprise JavaBeans™ Specification, v2.1, 2003. 493～499
10　Holzmann G J. The Model Checker SPIN. IEEE Trans on Software Engineering, 1997, 23(5): 279～295
11　Godefroid P, Holzmann G J. On the Verification of Temporal Properties. In: Proc PSTV'93, 1993. 109～124
12　Sun Microsystems Inc. J2EE Compatibility: http://java.sun.com/j2ee/compatibility.html