自适应重配置软件系统的运行时监控方法研究

唐 姗 李丽萍 谭文安
(上海第二工业大学计算机与信息学院 上海 201209)

摘 要 运行时监控技术作为实现自适应软件的一个重要研究内容,现已成为当前很多软件工程方法中用来提高软件产品可信性的一个重要设计原则。针对现有的很多软件监控方法常常将系统的监控逻辑与业务功能逻辑混杂在一起的问题,提出了一个需求模型驱动的、自适应重配置软件的运行时监控方法。以软件系统的目标模型及属性规约为基础,介绍了如何构建系统的监控模型、生成和编织监控代码,以及进行运行时诊断分析和自适应重配置调整。该方法通过采用独立于应用程序的外部单元来实现对运行时系统的监控、诊断和自适应重配置处理。这更利于系统的维护和管理,也更符合软件复用的思想。

关键词 目标模型,运行时监控,自适应,重配置
中图法分类号 TP311.5 **文献标识码** A

Research on Runtime Monitoring for Self-adaptive and Reconfigurable Software Systems

TANG Shan LI Li-ping TAN Wen-an
(School of Computer and Information, Shanghai Second Polytechnic University, Shanghai 201209, China)

Abstract Runtime monitoring is an important part of the study on self-adaptive software systems, and it is also an important design principle in building dependable software systems. However, most of the existing research works on runtime monitoring often mix the monitoring logic and the business logic. Different from these existing works, this paper proposed a requirement-model driven approach for self-adaptive software systems. Based on the goal model and the constraints specification, we illustrated how to define the monitoring model, how to deduce the specification of monitor from the goal model, how to generate and weave the monitoring codes automatically, how to diagnose and reconfigure the target system at runtime. This approach implements these tasks by separating the application code and the monitoring code, which is essential for facilitating software maintenance and promoting software reuse.

Keywords Goal model, Runtime monitoring, Self-adaptation, Reconfiguration

1 引言

近年来,随着因特网向社会各角落的渗透式扩张,普适计算、网格计算等新型应用模式不断涌现,软件的规模日趋庞大,复杂性日益增加,使得软件系统变得越来越难以驾驭,这给软件系统的可信性带来了新的问题和挑战。尤其是当开放性成为了 Internet 上软件的主要特征之后,运行在 Internet 环境下的应用系统的高层业务目标、系统结构和运行环境等均会不断发生变化。传统的软件容错技术也越来越难以应对软件的这些变化性。在新的应用背景下,为了保证软件的运行行为及其结果符合相关涉众的预期(即目标与实现相一致),许多研究人员提出通过增强系统的自适应(self-adaptive)能力加以实现。以 IBM 提出的自治计算[1]为代表的自适应软件模型已得到了学术界和产业界的广泛关注。

软件运行时监控是实现自适应重配置目标的基础。软件运行时监控技术通过实时收集和分析目标系统的运行时状态和行为信息,来判断系统的运行时行为是否满足系统的属性规约,从而发现系统的缺陷、异常和一些 QoS(Quality of Service)状况,为软件系统的动态自适应调整和演化等活动提供决策依据。自运行时监控技术提出以来,学术界和产业界都给予了广泛的关注,并提出了很多有代表性的方法。但是现有的很多方法,都将系统的监控逻辑与业务功能逻辑混杂在一起,不仅重载,使得系统的开发和维护工作变得相当的复杂和容易出错,且不利于软件复用。针对这些问题,本文在现有方法的基础上,采用自治计算[1]的思想,提出一个需求模型驱动的运行时监控与诊断分析方法。本方法将系统的监控逻辑与业务功能逻辑的实现相分开,希望以一种不影响系统正常运行的方式获取系统和各个构件的运行状态信息。与现有方法不同的是,本方法通过采用独立于应用程序的外部单元(监控单元、诊断单元以及自适应重配置修复单元)来实现对运行时系统的监控、诊断和自适应重配置处理。这更利于系统的维护和管理,同时也更符合软件复用的思想。

到稿日期:2013-01-23 返修日期:2013-09-03 本文受国家自然科学基金(61272036),2012年度上海高校青年教师培养资助计划(ZZegd12008),上海市教育委员会科研创新项目(11YZ251)资助。

唐 姗(1982—),女,博士,讲师,主要研究方向为软件工程、自适应软件、可信计算等,E-mail:tangshan@fudan.edu.cn;李丽萍(1976—),女,博士,副教授,主要研究方向为软件工程、形式化方法;谭文安(1965—),男,博士,教授,主要研究方向为软件工程、软件协同。

本文第 2 节介绍了本方法的基本框架；第 3 节介绍了如何对软件系统的需求及属性规约进行建模；第 4 节给出了一个基于构件的监控模型；第 5 节介绍了基于 AOP 的监控代码生成和编织过程；第 6 节介绍了基于目标模型的运行时诊断方法；第 7 节介绍了如何制定重配置策略，对系统进行自适应调整；第 8 节讨论了相关工作及本方法的优点；最后是结束语。

2 方法概述

运行时监控系统是一个观测系统，用于判断目标系统的执行行为是否与给定的需求相一致。传统的监控系统只负责检测系统的执行行为正确与否，不具备动态的容错处理能力。本文所提的方法在此基础上做了进一步的扩展，既能实现对软件系统的运行时监控与诊断，还能使其具备运行时容错的能力。本方法的基本研究思路是：以系统的需求模型为基础，对系统进行运行时监控和推理诊断（发现违反需求规约的反例），并生成和执行自适应重配置方案（减少系统运行时失效和错误的发生），实现从运行时控制的角度保证系统的可信性。本方法的整体框架如图 1 所示。

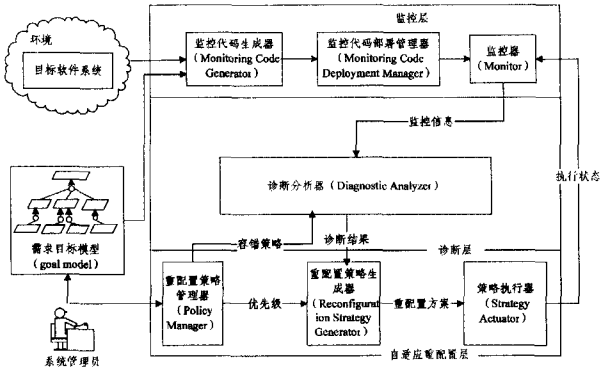

图 1 基于目标模型的运行时监控、诊断、自适应重配置概念框架

本方法的研究工作主要包括：(1)系统的需求及属性规约描述；(2)定义目标系统中要监控的对象；(3)生成和编织监控代码；(4)诊断目标系统的执行状况；(5)必要时的自适应重配置处理。

3 系统需求及属性规约描述

系统需求及属性规约是进行运行时监控和诊断的基础。目标是指与软件系统开发有关的所有涉众(stakeholders)期望该系统在现在或将来所能完成的任务以及任务需要完成的程度[2]。面向目标的需求建模与分析方法是当今一个热门的需求工程方法，它可以通过系统化地对系统目标进行抽象、泛化和求精等操作来保证最终所得的需求模型的一致性，并且支持共性和可变性分析，同时它还具有严格的形式化描述基础，具备逻辑推理的能力。因此，本文采用面向目标的需求建模方法对系统的需求进行描述，并在系统的目标模型中采用线性时序逻辑(LTL)来描述系统的属性规约。LTL 可以方便准确地描述并发系统的重要性质，如安全性(safety)和活性(liveness)。安全性用于说明“系统需求规约中不期望发生的事情永远都不会发生”，而活性则用于说明“需求规约中期望发生的事情最终会发生”。

例如，针对网上购物系统的“在线支付”功能，我们采用目标模型和 LTL 对其进行需求建模和属性规约，如图 2 所示。

Goal	*Achieve*[On-line Payment]
Concerns	Order_Num, Total_Cost, CreditCard_Num, Pay_Result...
Refines	*Achieve*[On-line Shopping]
RefinedTo	*Achieve*[Payment Option Selection], *Achieve*[Avoid Wrong Deduction], *Achieve*[Authentication], *Achieve*[Deduction Notification],
InformalDef	如果是VIP客户，系统必须先按VIP客户可享受的折扣给订单打折，再执行扣款操作；
FormalDef	$\forall u \in User,\ o \in Order\ \Box(VIP(u,o) \Rightarrow (\neg Pay(u,o) \cup Discount(u,o)))$

图 2 系统需求及属性规约描述实例

其中，目标“On-line Payment”表示要实现“在线支付”功能的目标，FormalDef 字段的描述内容 $\forall u \in User, o \in Order$：$\Box(VIP(u,o) \Rightarrow (\neg Pay(u,o) \cup Discount(u,o)))$ 表示系统实现该目标必须满足的约束条件，即为 VIP 客户进行订单结算时，每次必须先执行打折操作，才能执行付款操作。

接下来便可以目标模型为基础，推导出相应的监控器的形式化描述，进而自动生成相应的监控代码。

3.1 运行时监控器的推导

受模型检测技术的启发（通过监测反例来发现目标系统异常），我们研究如何根据系统的属性规约来构造目标系统的监控器。基于 LTL 的运行时监控器的推导过程如下：

(1)首先，根据目标模型中属性规约的 LTL 描述公式（表示系统应该满足的性质），生成违背系统期望属性的反例。我们可以基于 KAOS 方法框架中定义的各种故障分析模式来准确快速地推导系统目标的反例描述。比如，表 1 描述了文献[3]为形如“$R \Rightarrow \Diamond S$”的目标而定义的故障分析模式；

(2)然后，在生成的反例描述的基础上，为 LTL 公式中的谓词指定相应的监控主体(Agent)以及一些触发异常的条件信息，由此生成监控器的基于 LTL 的形式化描述；

(3)最后，将监控器的 LTL 描述转换成 Büchi 自动机的表示形式。Büchi 自动机是一个五元组 $\langle S, A, \Delta, q_0, F \rangle$，其中，$S$ 是有限的状态集合，A 是有限的字母集合，$\Delta \subseteq S \times A \times S$ 是一个由字母标识的迁移关系，$q_0 \in S$ 是初始状态，$F \subseteq S$ 是可接受状态集合。目前这方面的转换算法很多，我们采用经典的从 LTL 到 Büchi 自动机的自动转换算法[4]，生成监控器的有限状态机(FSM)的描述形式。

表 1 目标 $R \Rightarrow \Diamond S$ 的故障分析模式

	Assertion	Subobstacle
1-step regress	$S \Rightarrow P$	$\Diamond[R \wedge \Box \rightarrow P]$
	$S \Rightarrow P$	$\Diamond[R \wedge (\rightarrow S \cup \Box \rightarrow P)]$
starvation	$S \Rightarrow P$	$\Diamond[R \wedge \Box(\rightarrow S \cup \rightarrow P)]$
missing source	$R \wedge \Diamond S \Rightarrow P$	$\Diamond[R \wedge \rightarrow P]$
non-persistence	$R \wedge \Diamond S \Rightarrow P \omega S$	$\Diamond[R \wedge \rightarrow S \cup (\rightarrow P \wedge \rightarrow S)]$
non-persistence	$R \wedge \Diamond S \Rightarrow P \omega (P \wedge S)$	$\Diamond[R \wedge \rightarrow S \cup \rightarrow P]$
milestone	$R \wedge \Diamond S \Rightarrow \rightarrow S \omega M$	$\Diamond[R \wedge \Box \rightarrow M]$
blocking	$B \Rightarrow \Box \rightarrow S$	$\Diamond[R \wedge (\rightarrow S \cup B)]$
substitution	$S' \Rightarrow \Box \rightarrow S \wedge \blacksquare \rightarrow S$	$\Diamond[R \wedge \Diamond S']$
strengthening	$R \wedge \Diamond S \Rightarrow \Diamond[P \wedge (P \omega S)]$	$\Diamond[R \wedge \Box \rightarrow P]$
starvation	$R \wedge \Diamond S \Rightarrow \Diamond[P \wedge (P \omega S)]$	$\Diamond[R \wedge (\rightarrow S \cup \Box \rightarrow P)]$
	$R \wedge \Diamond S \Rightarrow \Diamond[P \wedge (P \omega S)]$	$\Diamond[R \wedge (\rightarrow S \cup (\rightarrow S \wedge \Box \rightarrow P))]$

4 监控对象定义

我们研究的是基于构件的层次化软件系统。在系统的运行过程中，构件之间通过接口进行交互活动，这些活动组成了系统的运行时体系结构行为。每个构件的运行时状态信息是我们判断整个系统的运行状态的基本依据。所以，这里以构

件作为基本研究对象来定义监控对象。

结合对现有监控方法的综合考虑，本文归纳出一个可扩展的基于构件的监控模型（如图 3 所示），以指导系统开发人员设计各种类型的监控探针（probe）。根据该监控模型可知，在监控运行时系统中的构件实例时，需监控的对象主要分为如下几类：构件间的消息通信、上下文环境、方法调用事件、属性变量值、运行平台的计算资源。

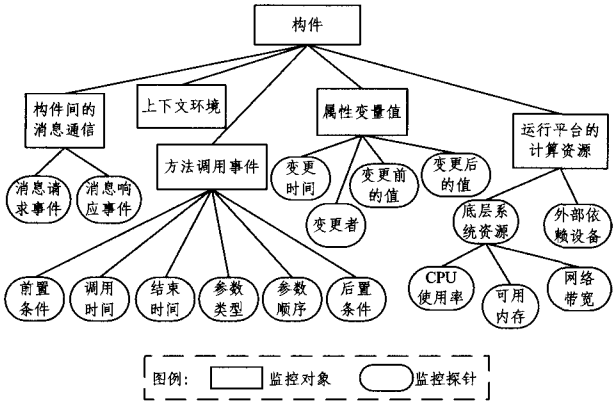

图 3　基于构件的监控模型

对构件间的消息通信事件的监控可分为消息请求监控和消息响应监控两种，这两类监控信息主要用于分析运行时系统的服务质量：（1）由于当前的很多软件系统运行在开放的 Internet 平台上，服务提供方可同时响应多个客户端构件的请求服务，因此并发连接的客户端数目、客户端构件的消息请求频率以及无效的或恶意的消息请求等与消息请求事件相关的因素都会影响到系统的执行情况。（2）消息响应事件是与系统服务质量属性[5,6]的评估最相关的事件之一。一般来说，我们通过对消息响应时间的监控来直接评估系统的实时性（Real time）、可用性（Availability）、有效性（Efficiency）和正确性（Correctness）等终端用户能够直观感受得到的服务质量属性。同时，基于消息响应事件的监控信息，还可以进一步推导出系统其它方面的质量属性，如系统失效的平均次数、系统出错的平均次数、系统的平均响应时间、系统在某一段时间内的响应次数等。

上下文（Context）是一个内涵不断演化的概念，在不同的应用研究领域中，人们对上下文的理解和定义是不同的。这里的上下文环境是指构件实例间的依赖关系，以及创建构件实例必须满足的约束条件。比如，在“网上宠物商店”系统中，我们规定“当某类商品的库存不够时（Cont1：Inventory ＜ Threshold_I）”，系统转换到“预定模式下”运行，创建“执行预定业务的构件实例”，那么 Cont1 就是该构件实例在这个场景下执行的上下文环境信息。上下文监控信息对于构件实例的生命周期管理和判断系统运行时行为的正确性都十分重要。

构件接口中的方法调用事件的监控信息通常用于检测系统运行时行为的正确性。为了更好地满足运行时监控和诊断的需要，我们通常需要监控被调用方法的“前/后置条件”、“开始执行的时间（调用时间）”、“执行结束的时间”、“方法参数”等信息来实现对方法调用事件的监控。（1）方法执行的前/后置条件是指在方法执行前/后，系统需满足的状态条件。前/后置条件的监控信息通常用于帮助系统实现安全性（比如为系统的授权认证、访问控制等操作提供决策依据）、正确性等目标。比如我们规定系统在进行自适应重配置操作前，必须进入某种受限模式（没有活动会话、某些关键服务被挂起等），才能执行一些关键性的自适应重配置操作，“进入到受限模式”就是系统执行自适应重配置操作的“前置条件”。（2）监控方法的调用时间和结束时间可用来检测系统的运行时行为是否满足需求目标模型中对应目标的“承诺时限”，还可用来判断方法的执行顺序是否符合需求规约中规定的约束条件。比如在“网上宠物商店”这个系统的需求规约中，我们规定客户端构件只有先执行了订购（“opOrder”）操作，才能开始执行付款（“opPay”）操作，那么通过监控 opOrder 方法的结束时间和 opPay 方法的调用时间可判断相关处理流程的正确性。（3）监控方法参数对保证系统的正确执行也是很重要的，监控信息主要包括对参数类型和参数传递顺序的监控。比如，在“方法 opOrder”中有个正整数类型的参数 ItemNum（表示订单中含有的商品数目），如果在提交订单时，检测到该参数值为零，则抛出参数异常，及时终止系统执行错误的行为（假设在需求规约中，明确给出了一个约束条件：在用户提交订单时，订单中至少含有一件选购的商品）。

系统中各个构件实例的属性变量值直接反映了软件系统的运行时状态，它们的变化反映了系统状态的变化情况。要验证自适应系统变更行为的正确性和有效性就必须监控系统中的关键变量值的变更信息，包括：变量值的变更时间、实施变更的主体（变更者）以及变更前后的值。这样在系统出现自适应重配置错误/失效时，这些监控信息可帮助系统定位错误、追踪分析引起错误/失效产生的原因以及进行自恢复处理。

运行平台的计算资源包括系统运行的外部依赖设备（比如数据库）和底层系统资源（比如 CPU 使用率、可用内存、网络带宽等）。这些资源的变更情况将直接影响到系统的运行时行为和执行效率。如果外部依赖设备失效或是资源出现瓶颈，则会影响系统的可信性，甚至造成暂时的系统失效。对这些资源信息进行监控可以提前发现和预测系统失效的发生，从而及时作出预警，避免上述不利情况的发生。

从以上分析可知：将监控模型中的各个探针事件与系统需求模型中完成任务的各项活动、任务的前/后置条件、上下文环境等信息关联起来，便可在需求模型与系统实现之间建立起良好的追踪关系，为运行时监控和诊断系统的可信性提供有效的支持。

5　生成和编织监控代码

监控代码主要由探针组成。探针可以从目标软件系统中实时获取感兴趣的事件、运行状态以及上下文环境信息。将监测代码（探针）嵌入到目标程序的功能代码当中来实现对目标程序的监控，这通常也被我们称之为基于“编织”模式的监控实现方式。这种方法的最大优点是探针可以被自由地插入到目标程序中的任何位置，实现效率高，而且不受系统运行平台的限制。但传统的方法是将监测代码和目标程序捆绑在一起，因此不利于程序的维护和运行时的重配置。

为了尽量避免代码的纠结和分散，本文采用面向方面编程（AOP）技术来实现目标系统的监控逻辑，从而很好地实现了系统的业务逻辑与监控逻辑的分离，降低了模块间的耦合度，提高了系统的可操作性和可维护性。图 4 描述了基于

AOP 的监控代码生成和编织过程。整个过程可以概括为分离实现与织入融合。前者是指将不同的关注点进行分离实现，后者是指将分离实现的所有内容进行整合，以形成最终的目标系统。

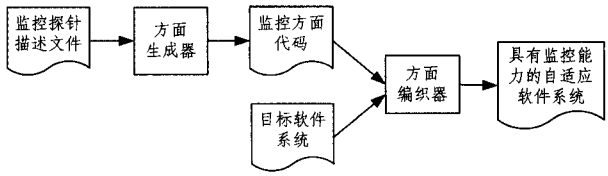

图 4　基于 AOP 的监控代码生成和编织过程

图 4 所示的整个过程主要分为如下 3 个步骤：

(1)首先根据在系统设计阶段所确定的系统的监控对象及监控需求，为相应的逻辑功能单元编写相应的探针描述文件。一条监控探针的描述信息主要包括探针类型、监控元素名称、监控属性值 3 个方面。其中，监控探针类型可根据第 4 节中的分析模型来具体指定，监控元素名称通常由开发人员根据监控对象来命名，而监控属性则可由一个链表来描述，用以存放多个可能的属性值。监控探针的 BNF 描述如下：Type 定义了可选的监控探针类型，主要包括构件间的通信消息 Message、上下文环境 Context、方法调用事件 Function、属性变量 Variable 和运行平台资源 Source 5 个关键字。

〈Probe〉::=〈Type〉〈Name〉{〈AttributeSet〉}

〈Type〉::=〈Message〉|〈Context〉|〈Function〉|〈Variable〉|〈Source〉

〈AttributeSet〉::=〈List of Attribute〉

(2)接下来“方面生成器”以监控探针描述文件作为输入，生成监控方面代码。在此需要根据目标编程语言，选择一款对应的 AOP 编译器，这里假设目标系统是采用 Java 语言来实现的，我们选择 AspectJ 来编译生成 AspectJ 监控方面代码。一个 AspectJ 方面类似于 OO 中的类，它主要封装了切入点(pointcut)和通知(advice)两类要素。

(3)最后由方面编织器 AspectJ 将 AspectJ 文件编织到目标软件系统中，形成一个具有监控能力的自适应软件系统。

6　基于目标模型的运行时诊断

一个系统是否可信，可通过比较和验证“由监控器传过来的监控信息”与“需求规约模型中所期望的系统属性信息”是否一致来判断。如果出现了不期望发生的事件/状态，则认为系统出现了错误；如果期望的事件/状态没有发生，则认为系统产生了失效。

近年来，很多文献采用人工智能领域里的命题逻辑的可满足性(SAT)技术来解决软件需求的诊断问题。SAT 问题是指对于一个命题逻辑公式 Φ，是否存在对其变元的一个真值赋值 μ，使 Φ 成立。如果存在一个真值赋值 μ，则称 Φ 得到了满足。那么我们只要将需求模型中的目标关系以及系统的运行时监控数据翻译成命题逻辑公式，便可利用 SAT 技术实现对可信需求的运行时诊断。

为了能够自动检测出系统目标和任务的错误/失效，需要首先采用形式化的方法来描述出系统期望/允许的目标和任务。这里引用文献[7]中提出的一组一阶逻辑规则来描述期望/允许的目标(goal)和任务(task)，假设每个目标实例之间可以用参数加以区别。

规则 1　对于一组参数值为 P 的根目标 G，如果其目标实例还未得到实现，则等到目标 G 被激活时，必须实现目标 G。这条规则描述了一个顶层目标应该在何时实现。其对应的一阶逻辑描述公式为：

$$goal(G) \wedge goal\_parameters(G,P) \wedge \neg\exists G_p\ s.t.\ goal(G_p) \wedge decomposed(G_p,G) \wedge \neg done(G,P) \wedge activation\_event(G,P,T) \wedge T \leqslant current\_time \rightarrow should\_do(G,P)$$

规则 2　一个期望实现的目标实例，也应该是系统允许发生的事件。其对应的一阶逻辑描述公式为：

$$should\_do(G,P) \rightarrow can\_do(G,P)$$

规则 3　如果目标 G 不是根目标，该目标只有满足如下 4 个条件，才允许发生：i)目标 G 是另一个可实现的目标 G_p 的子目标；ii)相应的上下文条件成立；iii)与父目标 G_p 的参数一致；iv)目标 G 还没有实现。其对应的一阶逻辑描述公式为：

$$goal(G) \wedge goal\_parameters(G,P) \wedge \neg done(G,P) \wedge \exists G_p\ s.t.\ goal(G_p) \wedge goal\_parameters(G_p,P_p) \wedge decomposed(G_p,G,Dec) \wedge can\_do(G_p,P_p) \wedge context\_cond(Dec) \wedge \forall p \in P\ s.t.\ (\exists p_p \in P_p\ s.t.\ name(p,n) \wedge name(p_p,n)), value(p,v) \wedge value(p_p,v) \rightarrow can\_do(G,P)$$

规则 4　一个任务除了满足规则 3 中定义的子目标需满足的条件外，还需满足另外两个条件才允许发生：i)必须满足任务可执行的前置条件；ii)与目标通过“手段-目的(means_end)”关系相连接。

$$task(T) \wedge task\_parameters(T,P) \wedge pre\_cond(T,P) \wedge \neg done(T,P) \wedge \exists G\ s.t.\ goal(G) \wedge goal\_parameters(G_p,P_p) \wedge means\_end(G,T,Dec) \wedge context\_cond(Dec) \wedge can\_do(G,P_p) \wedge \forall p \in P\ s.t.\ (\exists p_p \in P_p\ s.t.\ name(p,n) \wedge name(p_p,n)), value(p,v) \wedge value(p_p,v) \rightarrow can\_do(T,P)$$

接下来，在基于规则 1－规则 4 识别出的期望和允许的目标基础上，与所监控到的行为进行比较实现诊断。

7　自适应重配置调整

自适应重配置是指系统自治地从一种配置切换到另一种更好的配置的能力。自适应重配置层的主要功能是当系统出现错误/失效时，根据事先制定好的系统重配置规则，规划、部署和执行重配置策略，通过这种自修复机制来实现系统的自适应容错能力。与硬编码的方式相比，这种方式可极大地方便系统在运行时根据需求进行在线更新和演化。

在容错决策方面，根据所针对的不同问题存在确定性决策和不确定性决策两种情况。确定性决策的决策依据和决策结果都是确定、明确、可预先设计的，例如与外部服务失效相关的容错决策。不确定性决策则往往与多目标的权衡以及容错效果的不确定性相关，例如某些非功能性目标不满足时的动态调节以及当可用资源不足时所做出的目标取舍分析等。对于确定性决策，相应的决策方法主要以系统目标模型、环境模型以及静态建模的容错策略为依据，例如根据外部服务依

赖性关系确定实现构件的替换决策。对于不确定性决策，相应的决策方法以基于反馈的控制优化和预测方法为主，即通过自我学习和优化实现动态的容错策略，在此过程中反馈信息的获取(对容错效果的度量)和反馈模型的建立是关键问题。

在重配置策略建模阶段，系统管理员首先需要针对系统可能出现的各种错误/失效定义相应的重配置容错策略，描述软件系统在运行时的自适应规则，规则的格式为："If Condition Then Action"，表示当条件"Condition"为真时执行动作"Action"。一个条件 *con* 可以用一个三元组〈*e*,*s*,*cr*〉来表示，其中 *e* 是上下文事件，*s* 是上下文状态，*cr* 是对上下文的值的约束。一个动作 *act* 可以用一个四元组〈*exe*,*op*,*in*,*re*〉来表示，其中 *exe* 是动作的执行者，*op* 是完成该动作要执行的操作，*in* 是输入参数，*re* 是动作执行成功与否的标志。一条策略 *p* 可以用一个四元组〈*ID*, *pr*,*con*,*act*〉来表示，其中 *ID* 是策略 *p* 的唯一标识，*pr* 是优先级，*con* 是策略 *p* 的触发条件，*act* 是 *p* 的动作。

各条策略的优先级不同，当系统运行时，同一事件触发了多条策略，则优先执行优先级高的策略。策略的优先级也可以在系统运行时动态调整。策略的 BNF 定义形式如图 5 所示。

```
<Policy> ::= <ID> <Priority> <Condition><List>
<Priority> ::= <High> <Medium>|<Low>
<Condition> ::= <ContextEvent> <ContextState> <CompositeRestriction>
<ContextEvent> ::= <Primitive ContextEvent>|<CompositeContextEvent>
<PrimitiveContextEvent> ::= <EventType> <Supplier> <Attribute>
<EventType> ::= "PrimitiveEvent"
<CompositeContextEvent> ::= <EventOperator> <PrimitiveContextEvent>
<EventOperator> ::= "AND" | "OR" | "NOT" | "UNTIL"
<CompositeRestriction> ::= <CompositeOperator> <CompositeRestriction>
<CompositeRestriction> ::= <CompositeOperator> <Restriction>
<CompositeOperator> ::= "AND" | "OR" | "NOT"
<Restriction> ::= <Supplier> <Attribute> <Relation>
<Relation> ::= <RelationOperator> <Value>
<RelationOperator> ::= <"Over">|<"Below">
<Action> ::= <Executor> <Operator> {<ParameterSet>}
<ParameterSet> ::= <List of Parameter>
```

图 5　策略的 BNF 定义形式

```
<Policy ID = "SaleModeAdjust" Priority = "Medium">
<Condition>
  <ContextEvent EventType = "PrimitveEvent">
    <Supplier>Stock_sensor</Supplier>
    <Attribute>stock</Attribute>
  </ContextEvent>
  <ContextState Supplier = "Stock_Shortage_Lamp" Attribute = "switch_state"></ContextState>
  <CompositeRestriction CompositeOperator = "AND">
    <Restriction Supplier = "Stock_demo" Attribute = "stock">
      <Relation>
        <RelationOperator> Below</RelationOperator>
        <Value>1</Value>
      </Relation>
    </Restriction>
    <Restriction Supplier = "Stock_Shortage_Lamp" Attribute = "switch_state">
      <Relation>
        <RelationOperator> Is</RelationOperator>
        <Value>True</Value>
      </Relation>
    </Restriction>
  </CompositeRestriction>
</Condition>
<Action>
  <Executor>Sale_Mode_Adjuster</Executor>
  <Operation>Book</Operation>
</Action>
</Policy>
```

图 6　自适应策略的 XML 描述实例片段

以网上宠物商店为例，针对实时监控到的库存数据，我们制定如下一条自适应策略：当商品库存足够时，系统默认的销售模式是"直销模式"，如果商品的库存数低于 1 个，则将销售模式改为"预订模式"。该自适应策略的 XML 描述如图 6 所示。这条策略的 ID 为"SaleModeAdjust"，优先级为"Medium"，该策略由上下文原子事件触发，该原子事件通过传感器"Stock_sensor"来获取变量"stock"(库存)的变化情况，同时通过传感器"Stock_Shortage_Lamp"获取"switch_state"("紧缺指示开关")的上下文状态。当两个约束条件都为真时，则触发执行这条自适应调整策略。即，当"stock"的值小于 1，而且"switch_state"的值为真时，由"Sale_Mode_Adjuster"将销售模式切换为"Book"(预订)模式。

系统管理员定义好自适应容错策略之后，将这些策略规则加载到重配置策略管理器，以供图 1 中的"重配置策略生成器"决策、生成重配置方案之用。"策略执行器"则执行由"重配置策略生成器"生成的重配置方案，同时监控器对策略执行器的执行结果及状态进行监控，以保证系统自适应调整的正确性和有效性，防止出现错误的重配置行为而给系统的正常运行带来副作用，甚至造成危害。

8　相关工作对比分析

尽管运行时监控技术已出现了三十多年，但随着软件系统的规模和复杂性的不断增加，以及应用领域的不断扩大，运行时监控与诊断技术仍然受到学术界和产业界越来越多的关注。比较有代表性的工作有：文献[8]提出了一个基于监控和检测的框架，用以验证运行时系统的正确性。Zheng Li 等人提出了面向 Web 服务交互的运行时监测和验证框架[9]。MOP[10]采用面向监控的编程方法将运行时的监控行为作为设计软件的一项基本准则。该框架能根据开发人员预先定义的系统的期望属性自动生成监控器，并将其集成到目标系统中，以实时监控系统的动态行为。文献[11]提出了一个运行时监控框架，该框架支持应用程序开发人员定义系统行为的正确性，并可将这些属性转换成有限状态机(FSM)。在系统运行时，通过检测用 BPEL 描述的 Web 服务的有限执行轨迹与这些 FSM 所描述的属性规约是否一致来发现系统异常。针对基于阈值的自适应方法所带来的被动延迟问题，文献[12]提出了一个基于统计学的监控方法，该方法可提前预测系统的异常行为，实现了及时对系统进行自适应调整的目标。文献[13]根据规约语言、监控机制和事件处理器，对运行时监控系统进行了系统化的分类讨论。Avgustinov 等人[14]将 AOP 技术应用到系统的运行时行为监测中，使用 Aspect 来观察系统中发生的被关注的事件。Bodden[15]开发了一种运行时行为监测工具，使用线性时序逻辑描述系统行为，然后基于 AspeetJ 的织入机制，将监控代码编织到目标系统的源代码中。在这些工作的基础上，本文提出了一个基于目标模型的运行时监控方法，研究对运行时系统的在线监控和诊断，希望以一种不影响系统正常运行的方式获取系统和各个构件的运行状态信息。以目标模型为依据，研究软件系统的动态检测方法，从而及时发现系统运行时的错误。与现有的方法不同，很多监控方法只有检测系统错误的能力，而本文的方法还具有自适应容错的能力。

与上述工作相比，本文的方法具有如下几个方面的优点：(1)与平台无关，更加通用；(2)是一种由目标模型驱动的方法，由于目标模型描述了满足系统目标的多种可能的实现方式以及相应的质量关注和其他约束条件，提供了系统构件与涉众需求之间的追踪机制，因此为自适应体系结构的运行时

监控、自适应决策和演化实施提供了良好的基础；(3)将监控逻辑与业务功能逻辑的实现相分开，通过采用独立于应用程序的外部单元(监控单元、诊断单元以及自适应重配置修复单元)来实现系统的运行时监控和诊断，更利于系统的维护和管理，同时也更符合软件复用的思想。

结束语 本文提出了一个面向自适应重配置的运行时监控方法。本方法与传统的模型检测方法的不同之处在于它是一种动态的监控和诊断方法。本方法以目标模型为基础，通过对系统运行时的功能性行为进行监测，诊断分析软件系统的行为是否满足需求规约，并根据分析结果对系统进行自适应调整，以保证系统运行的正确性，可有效地提高软件系统的可信性。本文的主要贡献是：从监控系统的监控事件的定义，到如何生成和编织监控代码，再到如何诊断分析监控数据，并响应监控结果，进行自适应调整，给出了一个运行时监控、诊断和自适应重配置的整体概念框架。我们将在此基础上进一步开展相关支撑工具的实现工作。

参 考 文 献

[1] Kephart J O, Chess D M. The Vision of Autonomic Computing [J]. Computer, 2003, 36(1): 41-50

[2] van Lamsweerde A. Goal-Oriented Requirements Engineering: A Guided Tour[C]// Proceedings RE'01, 5th IEEE International Symposium on Requirements Engineering. Toronto, 2001: 249-263

[3] van Lamsweerde A, Letier E. Handling Obstacles in Goal-oriented Requirements Engineering[J]. IEEE Transactions on Software Engineering, 2000, 26(10): 978-1005

[4] Clarke E M, Grumberg O, Peled D A. Model Checking[M]. The MIT Press, 2001

[5] Taher L, Basha R, Khatib H E. Qos Information & Computation (QoS-IC) Framework for QoS-based Discovery of Web services [J]. The European Journal for the Informatics Professional, 2005, 6(4)

[6] Mani A, Nagarajan A. Understanding quality of service for Web services [OL]. http://www-128. ibm. com/developerworks/webservices/library/ws-quality. html

[7] Dalpiaz F, Giorgini P, Mylopoulos J. An architecture for requirements-driven self-reconfiguration [C] // Proceedings, CAiSE, Volume 5565 of LNCS. Springer, 2009: 246-260

[8] Lee I, Ben-Abdallah H, Kannan S, et al. A monitoring and checking framework for run-time correctness assurance[C]// Proceedings of the 1998 Korea-U. S. Technical Conference on Strategic Technologies. 1998

[9] Li Zheng, Jin Yan, Han Jun. A Runtime Monitoring and Validation Framework for Web Service Interactions[C]// Proceedings of ASWEC'06 Proceedings of the Australian Software Engineering Conference. IEEE Computer Society, 2006: 70-79

[10] Chen Feng, Rosu G. MOP: An Efficient and Generic Runtime Verification Framework [R]. Technical report UIUCDCS-R-2007-2836. March 2007

[11] Simmonds J, Ben-David S, Chechik M. Monitoring and Recovery of Web Service Applications[C]// The Smart Internet 2010. Springer, 2010: 250-288

[12] Amin A, Colman A, Grunske L. Using Automated Control Charts for the Runtime Evaluation of QoS Attributes[C]// Proceedings of the 13ht IEEE International High Assurance Systems Engineering Symposium. IEEE Computer Society, 2011: 299-306

[13] Delgado N, Gates A Q, Roach S. A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools[J]. IEEE Transactions on Software Engineering, 2004, 30(12): 859-872

[14] Avgustinov P, Bodden E, Hajiyev E, et al. Aspect for trace monitoring[C]// Proc of Formal Approaches to Testing Systems and Runtime Verification(FATES/RV 2006). LNCS 4262, 2006: 20-39

[15] Bodden E. A lightweight LTL runtime verification tool for Java [C]// Proc of OOPSLA. 2004: 306-307

(上接第 180 页)

[4] Song X, Dou W, Chen J. A workflow framework for intelligent service composition[J]. Future Generation Computer Systems, 2011, 27(5): 627-636

[5] 方其庆，彭晓明，刘庆华，等. 结合 AI 规划和工作流的动态服务组合框架研究[J]. 计算机科学，2009，36(9)：110-114

[6] Narayanan S, McIlraith S A. Simulation, verification and automated composition of Web services[C]// Proceedings of the 11th international conference on World Wide Web. ACM, 2002: 77-88

[7] Evren D W, Wu D, Sirin E, et al. Automatic Web services composition using shop2[C]// Workshop on Planning for Web Services. 2003

[8] 刘峰，谭庆平，杨艳萍. 基于图论的 Web 服务合成算法[J]. 华中科技大学学报：自然科学版，2005，33：202-204

[9] 邓水光，吴健，李莹，等. 基于回溯树的 Web 服务自动组合[J]. 软件学报，2007，18(8)：1896-1910

[10] Cardoso J, Sheth A, Miller J, et al. Quality of service for workflows and Web service processes[J]. Web Semantics: Science, Services and Agents on the World Wide Web, 2004, 1(3): 281-308

[11] Dumas M, García-Bañuelos L, Polyvyanyy A, et al. Aggregate quality of service computation for composite services[J]. Service-Oriented Computing, 2010, 6470: 213-227

[12] Zheng H, Zhao W, Yang J, et al. Qos analysis for Web service compositions with complex structures[J]. IEEE Transactions on Service Computing, 2013, 6(3): 373-386

[13] Sirin E, Hendler J, Parsia B. Semi-automatic composition of Web services using semantic descriptions[C]// Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS. 2003

[14] Martin D, Burstein M, Hobbs J, et al. OWL-S: Semantic markup for Web services[J]. W3C Member submission, 2004, 22

[15] 范菁，刘韬，熊丽荣. 信用构件的刻面分类及检索方法研究[J]. 计算机系统应用，2008(6)

[16] 范菁，杨冰，熊丽荣. 基于功能语义的构件描述和检索研究[J]. 计算机系统应用，2009(4)：26-31

[17] 董天阳，李文杰，范菁，等. 业务流程驱动的森林仿真构件组装技术及应用研究[J]. 计算机科学，2012，39(9)：126-132