基于 SARSA 在线规划的软件体系结构自适应*

周 勇, 王 苹
(大连理工大学 软件学院, 辽宁 大连 116021)

摘 要: 提出基于 SARSA 算法的在线规划软件体系结构自适应方法,用来解决由于环境内在固有的不确定性、复杂性和不可预见性而产生的离线规划的局限性。在线规划方法指可以根据当前的环境状况自动选择行动的规划方法。结合 Robocode 的实例详细阐述了实现基于 SARSA 算法的在线规划方法的三个关键问题和过程策略;为解决自适应的状态和行动表述、适应度和可受理集合关键问题,提出了自适应在线规划的策略。最后用 Robocode 的坦克战斗实例,证明了基于 SARSA 在线规划软件体系结构自适应方法的可行性和有效性。

关键词: 自适应; 软件体系结构; 在线规划; SARSA
中图分类号: TP311 **文献标志码**: A **文章编号**: 1001-3695(2012)05-1756-05
doi:10.3969/j.issn.1001-3695.2012.05.041

SARSA-based on-line planning software architecture self-adaptation

ZHOU Yong, WANG Ping
(*School of Software Technology*, *Dalian University of Technology*, *Dalian Liaoning* 116021, *China*)

Abstract: This paper proposed SARSA-based on-line planning software architecture adaptive method to avoid the limits of off-line plamming made by the environments of the complex software systems with the inherent uncertainty, complexity and unpredictability. On-line planning methods could select the action automatically based on the current state of the environment. This paper described the key elements of on-line planning methods and process strategies. Finally used a Robocode instance named SARSABot as a case to fight with FireBot. The result of the experiment proves the feasibility and effectiveness of the SARSA based on-line approach.

Key words: self-adaptation; software architecture; on-line planning; SARSA

0 引言

随着面向对象技术、构件技术的发展,软件系统的复杂度和规模日益剧增,同时随着 Internet 的发展,软件系统处在更加开放和动态变化的环境中,软件系统变得难以管理和维护。为了降低软件管理和维护的成本,软件自适应性研究应运而生[1~3]。所谓软件自适应性就是软件能够感知环境的变化,并且能针对环境的变化而改变自身的行为,采取合适的行动,以适应环境资源、用户需求的变化以及系统的错误。目前软件自适应研究已经成为自主计算研究领域的一个重要分支[4]。

软件体系结构是由构件和连接件组成的,是一种用来理解系统目标的组织结构[5]。研究表明可以使用体系结构作为外部模型支持软件系统自适应过程[6~9]。体系结构模型可以从系统全局的角度刻画当前配置状态,有利于对系统级特征属性进行监控,而且也便于检查关键约束是否得到遵守,即运行时刻体系结构相关信息的改变可用来触发、驱动系统的自适应行为。另一方面,对系统的自适应动作可以在体系结构的抽象层面上得以表达,而且可以充分利用在体系结构方面已有的研究成果。

一般的自适应系统是具有反馈回路的闭合回路,在运行时根据开放环境或需求的动态变化而自动调整系统[4]。基于控制论中的控制环路,可以将适应过程分成监测、分析、规划和实施四个阶段[10]。其中规划阶段是一个行动选择问题,决定了系统如何适应环境的变化[11],是非常重要的一个阶段。

关于规划阶段已经有了很多的研究[7,12~14]。当前自适应规划机制可以分为基于策略和基于计划两类方法。基于策略的机制使用 ADL 描述语言描述不变式和策略之间的映射来实现体系结构自适应[7];基于计划的机制根据当前系统状态自动产生实现目标状态的行动路径[13]。这两种自适应规划机制都存在一个大的问题,即在系统运行之前必须定义出自适应的方式[14],换句话说,无论遇到什么样的具体环境情况,自适应系统必须执行开发人员已经选择的某一个或一系列行动。但是在现实的软件开发中,由于软件生存环境内在的不可预见性、复杂性和不确定性,以及用户需求纷繁复杂并动态变化,开发人员制定适应计划时目标常含糊不清,可预见的状态和行动也非常有限,测试环境很难准确描述真实世界[15]。因此开发者很难在系统部署之前定义明确的目标、状态、行动、奖励以及设计正确说明实际运行环境特点的测试环境。在自适应研究中融合强化学习的相关思想和算法是解决上述环境不确定等问题的有效方法。SARSA 已经在自主计算系统中得到应用[11],利用强化学习在细粒度上让系统具有自适应性。

收稿日期: 2011-09-20; **修回日期**: 2011-11-08 **基金项目**: 国家自然科学基金资助项目(50979010); 中央高校基本科研业务费专项资金资助项目(DUT10JR15)

作者简介: 周勇(1971-),男,山东安丘人,副教授,博士研究生,主要研究方向为高性能计算与数据流挖掘、软件体系结构可信与演化(kevinzh@dlut.edu.cn);王苹(1985-),女,硕士研究生,主要研究方向为软件体系结构、软件自适应性.

本文提出在规划阶段中,在体系结构层面使用 SARSA 算法的思想,研究一种具有学习能力的自适应规划机制,其中自适应行为的产生是根据环境状态在线规划的,此类自适应系统具有自学习的能力,开发者不用对环境有完备知识。

1　在线规划理论与 SARSA 算法

1.1　在线规划理论

在线规划是指可以根据当前的环境状况自动选择行动的规划方法。在线规划允许关于环境的信息不完整,当系统环境发生变化时,系统不需要人为干预,即能在环境中学习,并选择合适的行动,自动调整系统。在线规划能够应对动态变化的环境和用户变化的需求[14]。一般在线规划过程包括三个步骤。

a)选择。系统自动选择一个适合当前环境状况、并满足系统目标的行动。行动选择一般通过贪心算法[15]来实现,即选择当前最好的行动。但是这个解决办法可能会导致局部最优,所以系统能以一定的概率接受由贪心算法决定的最优选择,在某些时候拒绝最优选择而尝试一些别的行动。

b)评估。系统必须估计在选择阶段已选择行动的有效性。其关键是定义出数值的、能有效代表行动奖励的函数表达,以便实现奖励的积累和比较。奖励函数定义了对于自适应系统来说哪些是好的事情,哪些是不好的事情,如果前一步骤选择的行为带来了较低的奖励,则下次就可能会选择其他的行为。

c)积累。系统存储评估阶段得到的行动数值。系统必须调整已积累知识和新经验间的积累比例。如果系统过多地使用已经积累的经验知识,则会减慢实现最优规划的收敛速度。相反,若在积累阶段使用了太多的新经验,则可能导致规划无效。

1.2　SARSA 算法

SARSA 算法是强化学习中的一种在策略(on-policy)的时序差分算法[16],它由 Rummery 和 Naraajan 提出,Singh 证明了在策略算法的收敛性。SARSA 是在更新时用到五元组(s,a,r,s',a')而得名的。其中:s 表示当前状态;a 表示当前状态下选择的行动;r 是行动 a 的奖励;s'和 a'则分别是后续的状态和行动。SARSA 算法中行为值函数的迭代规则如式(1)所示。

$$Q(s_t,a_t)\leftarrow Q(s_t,a_t)+\alpha[r_{t+1}+\gamma Q(s_{t+1},a_{t+1})-Q(s_t,a_t)] \quad (1)$$

其中:Q 代表 Q 值;s 为状态;a 为行动;r 为奖励函数值;γ 为折扣因子;α 是步长参数,其中 $0\leqslant\alpha\leqslant 1$ 和 $0<\gamma\leqslant 1$。

SARSA 算法流程大致如下[15]:

a)初始化 $Q(s,a)$,比如可以将每一对状态—行动 Q 值设置为 0。

b)在状态 s_t 观察行为集合 $A(s)$,根据此状态的行为函数值 $Q(s_t,a)$,按照一定的策略 π 选择行为 a_t。例如可以采用 ε-greedy 算法选择行动,这个策略在大部分时候选择具有最大 Q 值的行动,但是以 ε 的概率随机选择行动[15]。

c)执行行为 a_t,观察所得到的奖励函数值和下一时刻的状态 s_{t+1},按照步骤 b)中的策略 π 得到 $Q(s_{t+1},a_{t+1})$,根据式(1)更新 $Q(s_t,a_t)$。

d)$s_t\leqslant s_{t+1}$,判断状态是否为目标状态,当访问到目标状态时,结束一次迭代循环,否则开始新的迭代循环。

2　基于 SARSA 算法的软件体系结构自适应在线规划方法

2.1　Robocode 可适应体系结构

Robocode 是 IBM 开发的 Java 战斗机器人平台,游戏者可以在平台上设计一个 Java 坦克,每个坦克有一个从战场上收集信息的感应器及一个执行动作的传动器。其规则和原理类似于现实中的坦克战斗,融合了机器学习、物理、数学等知识,是研究强化学习技术很好的平台。

软件体系结构应该足够灵活以使构件可以在运行时根据环境、系统的目标、项目干系人目的的改变而改变自己的行为[17]。可以使用 UML 来描述软件体系结构。动态配置体系结构需要设计合理、灵活的抽象体系结构,为此将 Robocode 坦克的体系结构表示成图 1 的 UML 包图的形式。这是一个黑板风格的体系结构包图,其优点在于可扩充性强、模块间耦合比较松散,能够支持体系结构的动态变化。图 1 中一共有五个包:learner-robot package 实现机器人的学习;infor package 包含敌人的信息和当前的学习状态;firing package 是开火控制的方案包,包含多种开火方案可供选择;targeting package 是对扫描到敌人的瞄准,为射击做好准备,包含了多种瞄准方案可供选择;maneuver package 实现机器人的移动控制,包含多种移动控制方案可供选择。

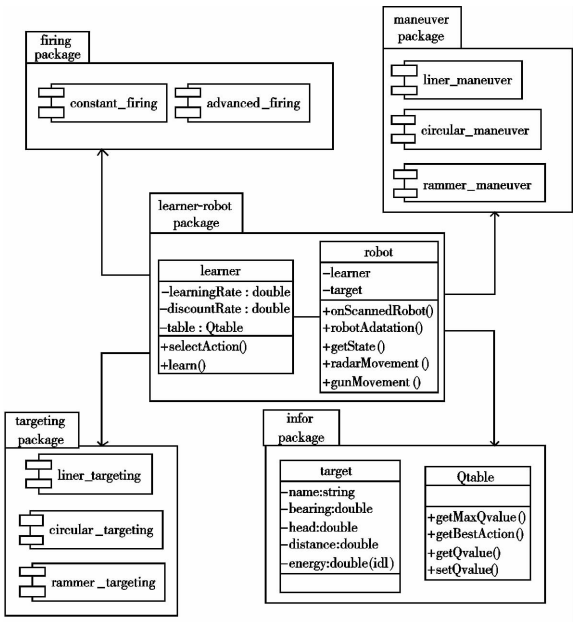

图1　Robocode坦克的体系结构

2.2　SARSA 实现自适应三个关键的问题

2.2.1　基于场景的体系结构状态和行动表述

状态和行动的表述直接关系到自适应方法的有效性,因为它们分别定义了系统的问题空间和解决方案空间,可以使用在软件需求工程研究中被广泛使用的基于目标[18]和基于场景的方法[19,20]来发现系统的体系结构状态和行动。采用这样的方法首先分层分析系统的目标,根据根本的系统目标逐渐得到每一层的子目标,形成一棵目标分析树(图 2)。目标树中每一个目标节点都应该有相应的场景。根据场景就可以整理出相应的状态和行动。Robocode 实例的目标、场景的具体描述如表 1 所示。状态是从场景的条件中获得的,一个条件代表了系统一个可能的状态。坦克所有的状态元素总结到了表 2 中。系统一共有七个状态,这些状态中有些其实是连续的值,为了能够分析状态将其离散化,如自身能量,当大于 50 时值为“高能

量”,小于等于 50 时值为“低能量”。

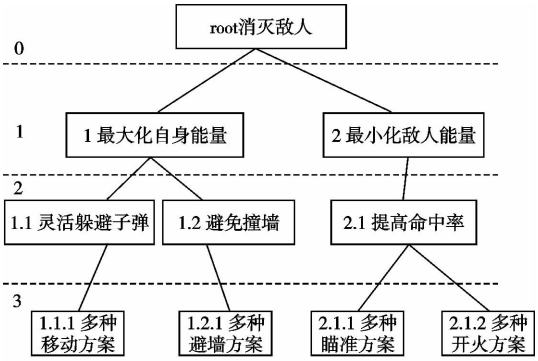

图2 坦克的目标树

表 1 基于场景的表述

目标	场景	条件	行动
root 消灭敌人	当敌人能量为 0 时,我方胜	敌人能量减少到 0	无
	我方拥有能量多获胜可能最大	能量最大	追击敌人
1 最大化自身能量	敌方发射子弹,躲过子弹	敌方发射子弹	躲过子弹
	撞墙,损失能量	撞到墙	避免撞墙
2 最小化敌人能量	射中敌人	射中敌人	维持现状
	火力大,损失能量多,但也可能获得更多能量	扫描敌人	调整火力
1.1 灵活躲避子弹	能量减少,改变移动方案,减少损失能量的可能	能量减少	改变移动方案
1.2 避免撞墙	撞墙,能量损失	撞到墙	避免撞墙
2.1 提高命中率	射击目标	敌人出现	发射子弹
	调整火力	射击敌人	调整火力
1.1.1 多种移动方案	被击中,采用直线移动,坦克继续往前移动	被子弹击中	直线移动
	被击中,若采用圆周运动,则按照圆周运动轨迹移动	被子弹击中	圆周移动
	若被子弹击中,若采用来回振荡运动,按照来回振荡移动	被子弹击中	来回振荡移动
1.2.1 多种避墙方案	在移动方案中加入避墙因子	撞到墙	加入避墙因子
2.1.1 多种瞄准方案	没有击中目标时,若用直线预测,使用直线瞄准构件	子弹没有击中目标	使用直线瞄准构件
	没有击中目标时,若用圆周预测,使用圆周瞄准构件	没有击中目标	使用圆周瞄准构件
	没有击中目标时,若用来回振荡预测,使用来回振荡瞄准构件	没有击中目标	使用来回振荡瞄准构件
2.1.2 多种开火方案	使用常量方案射击,火力不改变	射击敌人	使用常量开火构件
	若使用高级开火方案射击,根据敌我能量与距离调整开火火力	射击敌人	使用高级开火构件

表 2 状态元素

类型	取值	类型	取值
自身能量	{低, 高}	撞墙	{是,否}
敌人能量	{低, 高}	中弹	{是,否}
敌我距离	{近, 远}	命中敌人	{是,否}

从场景中确定的相应行动如表 1 所示,一个行动集是与一个条件相对应的。如果条件和行动集在系统设计阶段被确定,则被认为是离线规划,即静态的规划。本文方法的目标是在线规划的自适应,条件与行动集的映射关系应在运行时被系统学习发现。坦克的目标树如图 2 所示,在坦克大战中坦克的最根本目标是消灭敌人,取得战争的胜利。这个基本目标可以分成两个子目标:最大化自身的能量和最小化敌人的能量。在这个目标树中一共有十个目标节点。每一个节点都对应着相应的场景、条件和行动。

表 3 显示 Robocode 实例的行动元素类型和取值。一个行动元素,如移动路线控制为来回运动,表明了体系结构的变化。其中包括添加、删除、替换组件和改变架构的拓扑结构等。

表 3 行动元素

类型	取值
移动路线控制	{来回运动,直线运动,圆周运动}
瞄准控制	{针对圆周运动瞄准,针对来回运动瞄准,针对直线运动瞄准}
开火控制	{常量开火方案,高级开火方案}

在这个部分中所描述的状态和行动的发现过程,是通过从目标和场景中的可测量的变量识别状态和行动。由于目标和场景直接代表目标和用户经验,也展示系统如何影响环境,状态和行动的信息可以反映出系统应监测什么和环境存在变化时系统应如何重新配置体系结构。

2.2.2 适应度

使用 SARSA 算法的自适应系统把动态环境状态映射成相应的动作,通过不断尝试错误,从环境中得到奖惩的方法来自主学习不同状态下哪些动作具有最大的价值,从而发现或逼近能够得到最大奖励的策略。在这个过程中有一个对行动衡量的重要标准——适应度。适应度是指行动在特定环境状态下对目标的满足,代表了该系统选择的行动的奖励。适应度需要用数值表示,才能对其进行比较和积累。所以设计合理的适应度函数是至关重要的。本文利用在 2.2.1 节描述的目标和场景结构,其中目标是发现适应度函数的关键。

一般来说在自适应软件中,目标是分层次的,高层次的目标对于定义数值的函数来说过于抽象。因此,有必要找一个适当层次的目标来定义适应度函数。很难定义通用规则来表示如何选择合适的数值适应度函数,但是可以通过对分层目标的分析找到一些可以数值表述部分目标是否实现的函数,通过这些函数来构造最后的适应度函数,式(2)描述了整合适应度函数的方法:

$$r_t = f(t) = \sum w_i f_i(t) \tag{2}$$

其中:r_t 是在时间 t 的奖励,$f(t)$ 是适应度函数;w_i 是第 i 个适应度函数的权重;$f_i(t)$ 是第 i 个适应度函数。适应度函数是所有子适应度函数的加权求和,$\sum_i |\omega_i| = 1$。关于权重的设置也有很多研究,有均值自适应法、均值波动自适应法等。在例子中,目标分成了四层,目标层 1 中的两个子目标最大化自身能量和最小化敌方能量可以用数值化的函数表示,并且分别对它们赋予权重 0.2 和 0.8,则适应度函数可以表示为

$$r_t = 0.2 \times \Delta E_m(t) + (-1) \times 0.8 \times \Delta E_e(t) \tag{3}$$

其中:r_t 是 t 时刻的奖励;$\Delta E_m(t)$ 是 t 时刻和 $t-1$ 时刻坦克自身能量的变化;$\Delta E_e(t)$ 是 t 时刻和 $t-1$ 时刻敌人能量的变化。

2.2.3 可受理行动集合

强化学习使用算子减少搜索空间,即行动数量,可以降低系统的学习时间。$A(S)$ 指定观察到的状态可予受理的行动集合。例如,当一个移动机器人碰到了墙体,只需要移动路线控制相关的行动,不需要涉及开火控制和瞄准控制相关的行动。根据 2.2.1 节中场景分析得到了六个状态和八个行动,若每个状态对每一个行动都去学习,将花费大量时间,所以需要使用算子将与状态没有相关性的行动排除。例如 A(自身能量 =

低) = A(自身能量 = 高) = A(敌人能量 = 高) = A(敌人能量 = 低) = {常量开火方案,高级开火方案},A(敌我距离 = 近) = A(敌我距离 = 远) = {来回运动,圆周运动,直线运动,针对圆周运动瞄准,针对直线运动瞄准,针对来回运动瞄准},A(中弹 = 是) = {直线运动,来回运动,圆周运动},A(命中敌人 = 否) = {针对圆周运动,针对直线运动,针对来回运动}。

2.3　在线规划策略

本节介绍在线规划过程中实现前一节提出的三个关键问题的策略。这个策略包括检测、规划、执行、评估和学习五个阶段,如图 3 所示。该系统可以动态循环执行这些阶段以对体系结构进行自适应。

a)检测阶段。在检测阶段,该系统可监控当前环境下系统的状态。当检测到 2.2.1 节其中定义的状态时,采取自适应。持续不断地检测可能会导致系统性能下降,因此,关键是要监督实际上触发系统的适应的改变。已经定义的状态是适应的触发因子,因为它们描述系统需要适应的时刻,如果一个状态被检测到,那么系统将进行自适应。对目前该系统的架构进行重配置,将检测到得状态数据传递到规划阶段。

b)规划阶段。使用检测阶段确定的状态,该系统选择一个动作,以适应自己的状态。在这个阶段,系统采用贪心随机法(ε-greedy)的行动选择策略。这个行动选择策略大部分时候选择具有最大 Q 值的行动,以 ε 的概率随机选择行动。具体来说,在这个策略中,系统首先通过贪心算法在当前状态的可受理行动集合 $A(S)$ 中选择具有最大 Q 值的行动,作为一个参考行动。但是为了探索新的行动,再加入一个随机过程。由开发人员确定一个 ε 值,$0 \leqslant \varepsilon < 1$。在产生一个参考行动之后,系统会生成一个随机数 r,如果 $r > \varepsilon$,系统接受参考行动;如果 $r < \varepsilon$,系统拒绝接受这个参考行动,系统从受理行动集 $A(S)$ 中随机选择一个动作执行。在这种方式下,随机选择行动的方式防止系统陷入局部最优。

c)执行阶段。这一阶段执行前一阶段选择的动作。由于行动描述体系结构变化,如添加、删除、更换部件,重新配置体系结构、拓扑结构,系统必须具有架构操作设施。一旦重新配置完成后,系统通过使用重配置架构实现自己的功能。该系统保持运行直到遇到新的状态或者直到系统终止。

d)评估阶段。这一阶段是关系到 1.1 节说明的评价步骤,因为该阶段决定了评估的数值。执行阶段后,系统必须通过观察,从环境获得前一执行的奖励。系统不断观察 2.2.2 节定义的适应度函数值,这些值将用于计算所采取行动的奖励。

e)学习阶段。在这个阶段,系统积累从以前执行过程中获得的经验,通过使用评估阶段观察到的奖励,计算新的 Q 值。这个阶段直接使用 SARSA 方法。而 SARSA 的关键活动是更新 Q 值,式(1)对此更新过程进行了描述。在 SARSA 中使用实际的后续状态—行动值进行迭代计算状态和行动的 Q 值。在这种方式下,系统可以通过更新 $Q(s_t, a_t) = V$ 积累经验,其中 s_t 是检测到的状态,a_t 是在 s_t 时系统采取的行动,V 是在 s_t 下 a_t 的值,将在规划阶段用来指导选择当前最佳的行动。

总之通过不断执行检测、规划、执行、评估和学习这五个阶段,系统在运行时与变化的环境保持一致,适应环境,具有较好的性能。

3　实验结果

为了验证软件体系结构在线规划自适应方法的有效性,使用了强化学习与没有使用强化学习的坦克战斗。实验选择了 fire 机器人作为 learner 坦克机器人的对手进行战斗。将 Q-table 中每一个格的初始值设置为 0,分别设置三个参数 α、γ 和 ε 的取值为 $\alpha = 0.1$,$\gamma = 0.7$,$\varepsilon = 0.1$,奖励函数 $r_t = 0.2 \times \Delta E_m(t) + (-1) \times 0.8 \times \Delta E_e(t)$,运行战斗 100 轮,战斗场地大小为 800×600 pixel^2。

实验结果如表 4 所示,列出了每十轮战斗两个坦克分别的得分。使用 MATLAB 将实验结果转换成坐标曲线图,如图 4 所示。其中横坐标表示运行轮数,纵坐标表示得分,可以看出一开始 fireBot 占了上峰,但是随着运行轮数的增加,SARSABot 逐渐扭转局势,转败为胜,可以说明使用强化学习具有良好的性能,经过训练后的机器人具有在线自适应能力,通过一定时间的学习能够灵活地躲避子弹、避免撞墙与其他机器人相撞以及移动到一个优化的位置以便更好地进攻,最后战胜敌人。

表 4　实验结果

轮数	SARSABot 得分	fireBot 得分	轮数	SARSABot	fireBot
10	624	975	60	870	979
20	915	937	70	965	939
30	688	796	80	847	820
40	790	933	90	1 064	714
50	676	911	100	1 116	786

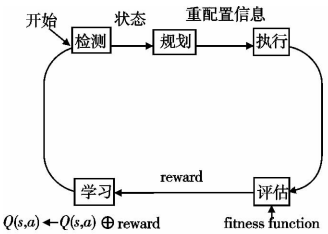

图3　在线规划的过程

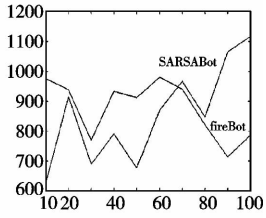

图4　SARSABot和fireBot战斗实验结果

4　结束语

本文指出了离线规划的限制性条件,并且分析了使用基于 SARSA 在线规划的软件体系结构自适应的必要性和可行性。结合 Robocode 实例,描述了使用基于 SARSA 的在线规划方法设计软件体系结构自适应系统,提供了支持在线规划的三个重要因素:状态和目标的表述、行动适应度以及可受理集合的演化过程。这个演化过程中,首先检测一个状态,其次规划一个行动过程,并执行这些行动,然后评估执行的效果,最后学习评估的结果。通过 Robocode 坦克实例证明了基于 SARSA 算法的软件体系结构自适应在线规划方法的有效性和可行性。但是将此方法运用到软件实际开发中还需要更加深入的研究。

参考文献:

[1] YANG Z, CHENG B H C, STIREWALT R E K, *et al*. An aspect-oriented approach to dynamic adaptation[C]//Proc of the 1st Workshop on Self-healing System. New York: ACM Press, 2002: 85-92.

[2] REILLY D, TALEB-BENDIAB A, LAWS A, *et al*. An instrumentation and control, based approach for distributed application management and adaptation[C]//Proc of the 1st Workshop on Self-healing Systems. New York: ACM Press, 2002: 61-66.

[3] GARLAN D, SCHMERL B. Model-based adaptation for self-healing systems[C]//Proc of the 1st Workshop on Self-healing System. New York: ACM Press, 2002: 27-32.

[4] SALEHIE M, TAHVILDARI L. Self-adaptive software: landscape and research challenges[J]. ACM Trans on Autonmous Adaptive Sys-

tems,2009,4(2):1-42.

[5] 张友生. 软件体系结构[M]. 北京:清华大学出版社,2004:18-24.

[6] OREIZY P, MEVIDOVIC N, TAYLOR R N. Architecture-based runtime software evolution[C]//Proc of the 20th International Conference on Software Engineering. Washington DC:IEEE Computer Society,1998:177-186.

[7] GARLAN D,CHENG Shang-wen,HUANG An-cheng. Rainbow:architecture-based self-adaptation with reusable infrastructure[J]. Computer,2004,37(10):46-54.

[8] GEORGAS J C,TAYLOR R N. Towards a knowledge-based approach to architectural adaptation management[C]//Proc of the 1st ACM SIGSOFT Workshop on Self-managed Systems. New York:ACM Press,2004:59-63.

[9] TAYLOR R N,MEDVIDOVIC N,OREIZY P. Architectural styles for runtime software adaptation[C]//Proc of the 3rd European Conference on software Architecture. Washington DC:IEEE Computer Society,2009:171-180.

[10] KEPHART J O,CHESS D M. The vision of autonomic computing [J]. Computer,2003,36(1):41-50.

[11] AMOUI M,SALEHIE M,MIRARAB S,*et al*. Adaptive action selection in autonomic software using reinforcement learning[C]//Proc of the 4th International Conference on Autonomic and Autonomous Systems. Washington DC:IEEE Conputer Society,2008:175-181.

[12] SALEHIE M,TAHVILDARI L. A weighted voting mechanism for action selection problem in self-adaptive software[C]//Proc of the 1st International Conference on Self-adaptive and Self-organizing Systems. Washington DC:IEEE Computer Society,2007:328-331.

[13] TAJALLI H,GARCIA J,EDWARDS G,*et al*. PLASMA:a plan-based layered architecture for software model-driven adaptation[C]//Proc of IEEE/ACM International Conference on Automated Software Engineering. New York:ACM Press,2010:467-476.

[14] KIM D,PARK S. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software[C]// Proc of ICSE Workshop on Software Engineering for Adaptive and Self-managing Systems. Washington DC: IEEE Computer Society, 2009:76-85.

[15] ELKHODARY A,ESFHANI N,MALEK S. FUSION:a framework for engineering self-tuning self-adaptive software systems[C]//Proc of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York:ACM Press,2010:7-16.

[16] SUTTON R S,BARTO A G. Reinforcement learning:an introduction [M]. Cambridge:MIT, 1998:150-185.

[17] TARVAINEN P. Adaptability evaluation of software architectures:a case study[C]//Porc of the 31st Annual International Computer Software and Applications Conference. Washington DC: IEEE Computer Society,2007: 579-586.

[18] HARMAN M,JONES B F. Search-based software engineering[J]. Information & Software Technology,2001,43(14):833-839.

[19] Van LAMSWEERDEA. Goal-oriented requirements engineering: a guided tour[C]//Proc of the 5th IEEE International Symposium on Requirements Engineering. 2001:249-262.

[20] SUTCLIFFE A G,MAIDEN N A M,MINOCHA S,*et al*. Supporting scenario-based requirements engineering[J]. IEEE Trans on Software Engineering,1998,24(12):1072-1088.