开放环境下自适应软件系统的运行机制与构造技术

毛新军[1),2)]，董孟高[3)]，齐治昌[1)]，尹俊文[1)]

[1)](国防科学技术大学 计算机学院计算机科学与技术系，长沙 湖南 中国 410073)

[2)](国防科学技术大学 计算机学院并行与分布处理重点实验室，长沙 湖南 中国 410073)

[3)](国防科学技术大学 装备综合保障技术重点实验室，长沙 湖南 中国 410073)

摘 要 由于环境变化的不确定性和不可预见性，开放环境下自适应系统的开发面临着以下二方面挑战：软件开发人员很难在设计阶段清晰地预测环境中各种可能的变化并准确地定义相应的自适应需求，系统大量的自适应决策需要由系统自身在运行时来完成。针对上述挑战，本文提出了一种基于软件 Agent 和组织抽象的方法来支持此类系统的开发和运行。该方法采用社会组织的思想来抽象自适应系统，描述和分析系统的自适应特征；设计了基于角色动态绑定的自适应运行机制，并借助于增强学习的手段来实现系统的在线自适应决策以应对不可预见的变化。论文介绍了基于学习和动态绑定机制的在线自适应决策算法，提出了支持开放环境下自适应软件系统的工程化开发手段，包括自适应软件模型和构造框架、结构化的过程，开发了相应的支撑软件环境 SADE+，并进行了案例分析以展示方法的有效性。

关键词 自适应; 组织; 动态绑定; 学习

Running Mechanism and Implementation Technique of Self-Adaptive Software in Open Environment

Mao Xinjun [1,2)], Dong Menggao [3)], Qi Zichang[1)], Yin Junwen [1)]

[1)] Department of Computer Science and Technology, College of Computer, National University of Defense Technology, Changsha, China, 410073

[2)] Science and Technology on Parallel and Distributed Processing Lab, College of Computer, National University of Defense Technology, Changsha, China, 410073

3) Lab of Science and Technology on Integrated Logistic Support, National University of Defense Technology, Changsha, China, 410073

Abstract Due to the uncertainty and unpredictability of environment changes, it is a great challenge to develop self-adaptive systems in open environment. First, it is difficult for developers to clearly predict various environment changes and precisely define self-adaptation requirements at design-time. Second, many of self-adaptation decisions should be made by system at run-time. In order to deal with the problems, the paper presents an approach that is based on software agent technology and organization metaphor to support the development and running of such systems. Our approach enables developer to describe self-adaptive systems and investigate self-adaptation according to the high-level organization abstractions. A self-adaptation mechanism called role dynamic binding is designed and on-line self-adaptation is achieved by introducing enforcement learning. The paper details the on-line self-adaptation decision algorithm that integrates dynamic binding mechanism with enforcement learning together. Especially, a general-purpose and systematics software

---

本课题得到国家自然科学基金(No. 61070034, 61379051)；教育部新世纪优秀人才计划(No. NCET-10-0898)和北航软件开发环境国家重点实验室开放课题(SKLSDE-2012KF-0X)资助..毛新军，男，1970年生，博士，教授，博士生导师，主要研究领域为软件工程、多Agent系统、自适应和自组织系统，E-mail: xjmao@nudt.edu.cn. 董孟高，男，1979年生，博士，讲师，主要研究领域为自适应系统和软件工程，E-mail:janson_mgd@163.com. 齐治昌，男，1942年生，教授，博士生导师，主要研究领域为软件工程，E-mail: qzc@nudt.edu.cn. 尹俊文，男，1969年生，硕士，副教授，硕士生导师，主要研究领域为软件工程，E-mail: jwyin@nudt.edu.cn.

engineering solution to developing such system is provided, including self-adaptive software model, implementation framework, structured process and supporting software environment SADE+. A case is studied to illustrate our approach and validate its effectiveness.

Key words Self-adaptation; Organization; Dynamic Binding; Learning

1 引言

当前越来越多的软件密集型系统部署在动态、开放和难控的环境中（如 Internet）[1]，并朝着规模超大的方向发展[2]。为了应对环境和系统自身的持续、动态变化，人们对系统的适应性提出了迫切的需求，以使得系统具有更好的灵活性、健壮性、可靠性、易变性、自优化性、可重配性等等[3]。自适应软件系统在互联网应用、柔性制造、国防军事、航空航天、企业计算等领域具有广泛的应用潜力。

自适应软件是指这样一类软件系统，它们可对自身进行调整以对发生在环境或者自身的变化做出响应[4]，以更好地满足系统的设计目标。这种调整可针对软件系统的不同方面（如属性、构件等等），在不同的层次（如数据、模块、体系结构等）上进行。借助于控制学理论，自适应系统的调整被视为是一个闭环反馈（closed-loop with feedback）过程，包含了四个方面的基本活动：监控（Monitor）以获取各种原始的变化数据、发现（Detect）以分析数据、决策（Decide）以确定如何进行调整和实施（Act）以执行调整[4]。

在软件工程和人工智能等领域，有关自适应软件系统的研究引起了人们的高度关注和重视[3][4]，研究内容涉及到自适应系统软件工程的诸多方面，包括：需求工程[3]、软件体系结构[5]、建模语言和方法学[6]、中间件[7]、构件技术[9]、语言设施[10][11]等等。由于软件系统的适应性给工程化理论和技术带来的挑战，近年来该领域研究的一个重要趋势是与其它相关领域的研究相融合，从而为自适应软件系统的开发提供基础理论和关键技术，如控制论[4]、智能 Agent 技术[6]等等。

目前自适应软件系统的开发大多采用自顶向下、设计优先的方式。该方式的特点是在设计阶段由软件开发人员显式地表示系统应该监控和发现哪些变化、针对这些变化系统该如何对自身进行什么样的调整，并采用策略（Strategy）[11]和规则（Rule）[12]的方式、借助于各种形式的语言设施来描述这些自适应需求及设计决策，如软件体系结构描述语言、自适应策略描述语言、面向方面程序设计语言等等。上述方法对于变化在开发阶段可确定和可预期的、调整可预先定义的自适应软件系统而言是有效，但是如果自适应软件系统所驻留的环境是开放的，那么现有的自适应系统软件工程技术将面临着一系列的挑战。

首先，设计阶段开发人员无法事先预知所有的自适应需求[3]。对于开放环境下的自适应系统而言，由于环境变化的不确定性和不可预见性，不可预期的事件随时可能发生，因而事先准确和完整地预期各种变化以及相应的自适应需求几乎是不可能的，由软件开发人员在设计阶段借助于各种自适应策略描述语言或者程序设计语言来定义自适应逻辑不可行。例如欲建设一个可信的自适应软件系统，它可针对网络攻击行为进行适应性调整，以使得系统具有更好的健壮性和灵活性。显然开发人员不可能预知各种可能的网络攻击方式和情况。其次，自适应系统通常需要持续性的运行。现有的自适应软件通常采用离线（off-line）决策的方法，即根据开发者所提供的预定义变化描述和良定义的自适应逻辑来实施调整。一旦自适应需求发生了变化（如出现了非预期的环境变化并需要对该变化作出自适应的响应），那么系统的自适应逻辑需要重新定义，系统的运行需要被终止并重新加载。对于开放环境下的自适应系统而言，变化的不可预知性将可能导致系统运行的经常性终止。

为了解决上述问题和挑战，开放环境下自适应系统的软件工程技术需要实现二方面的转变，即将原先由开发人员在设计阶段给出的自适应逻辑延迟到由系统自身在运行阶段来完成，自适应决策需要由离线方式转化为在线方式。为此，本文提出了一种基于软件 Agent 和组织抽象的方法来支持此类系统的开发和运行。该方法采用社会组织的思想来抽象和描述自适应系统，设计自适应运行机制，借助于增强学习的手段来应对不可预见的变化、实现系统运行时的自适应决策。论文剩余章节组织如下：第二节定义了基于组织抽象的自适应系统模型，在此基础上设计了角色动态绑定的自适应机制，第三节提出了基于学习和动态绑定的在线自适

应决策方法，第四节介绍了开放环境下自适应系统的工程化开发方法，并开展了案例分析。第五节介绍和对比分析了相关研究工作，最后第六节总结全文工作并展望了下一步的研究。

2 自适应系统的抽象模型及运行机制

当前越来越多的自适应软件系统实际上是现实社会系统在计算机世界中的映射，许多自适应软件系统具有显式的社会组织特征。组织学和社会学可为自适应软件系统的研究提供高层的抽象和概念（如角色、组织、规则等），它们直观、易于理解、更加贴近现实世界，有助于理解和分析“为什么(Why)”、“谁(Who)”、“如何(How)”以及针对什么样的对象(“What”)进行自适应，进而简化和控制自适应系统的开发。

图 1 描述了基于 Agent 和组织抽象的自适应系统元模型。一个自适应系统被为视为一个社会化的组织，组织中的个体被抽象为 Agent，组织及其 Agent 均驻留在特定的环境中，因而受环境变化的影响。组织拥有一组角色，它们封装了组织中的行为和资源。组织中的个体 Agent 通过扮演不同的角色从而来展示其在组织中不同的地位和行为。

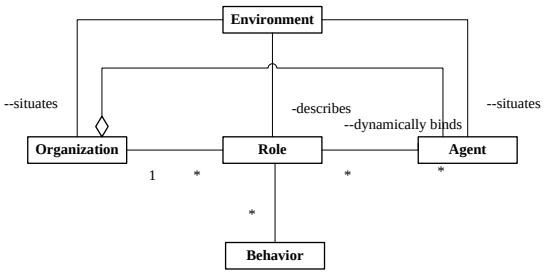

图 1 基于 Agent 和组织抽象的自适应系统元模型

■ **组织**（Organization）是指在特定上下文中一组具有共同目标、相互交互的 Agent 集合，组织的上下文定义了组织所驻留的环境。对于具有自适应能力的系统而言，环境变化将对组织的要素、结构、行为等产生影响。

■ **Agent** 是指驻留在特定环境下能够自主、灵活地执行动作以满足设计目标的行为实体。Agent 通过扮演组织中的角色成为组织中的成员，获得组织中的地位，拥有相应的资源和行为。对于某些 Agent 而言，驻留环境以及自身内部的变化将促使 Agent 对其自身的结构和行为进行调整，该类 Agent 被称为自适应 Agent（Self-adaptive Agent，简述为 SA）。

■ **角色**（Role）是对 Agent 在组织上下文中所展示的环境、行为及资源的抽象表示，反映了 Agent 在组织中的地位。

■ **环境**(Environment)是对哪些对组织或者 Agent 的行为或者调整产生影响的要素的抽象。环境的变化可抽象为环境中相关事件的发生。对于开放环境而言，环境变化往往是动态、不确定、无法事先预知的。

MAS 组织中的 Agent 能够随着驻留环境及其自身的变化动态地调整它在组织中所扮演的角色，进而“进入”或者“退出”某个组织，或者改变它在组织中的“地位”(Position)，反映了它对环境和自身变化的某种适应性。Agent 对角色的调整可通过以下四个基本的自适应原子操作来完成：“Join”（加入）、“Quit”（退出）、“Suspend”（挂起）、“Resume”(恢复)。一旦 Agent 通过执行这些原子操作来改变其角色，那么 Agent 的内部资源、属性和行为等也随之发生调整。本文将上述自适应机制称为动态绑定机制（见图 2）。

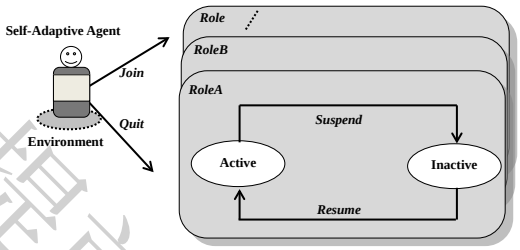

图 2 基于角色动态绑定的自适应机制

■ **“Join”**：Agent 可执行自适应动作“Join”以加入组织中的角色，获得该角色所定义的结构和行为特性，进而改变 Agent 在组织中的地位，如成为组织中的一员或者改变其在组织中所扮演的角色。一旦 Agent 加入某个角色，我们称 Agent 绑定了该角色。随之该角色所定义的结构和行为将对 Agent 实施约束和产生影响。

■ **“Quit”**：Agent 可执行自适应动作“Quit”以退出组织中的角色，失去该角色所定义的结构和行为特性，进而改变 Agent 在组织中的地位，如不再是组织中的一员或者改变其在组织中所扮演的角色。

Agent 所绑定的角色具有以下二种不同的状态：活跃（Active）状态和非活跃（Inactive）状态，并且在任何时刻只能处于其中的一种状态。

■ **“Suspend”**：Agent 可执行自适应动作“Suspend”以将所绑定角色的状态从“活跃”调整为

“非活跃”。当 Agent 所绑定的角色处于非活跃状态时，它将不再约束 Agent 的运行以及指导 Agent 的运行，但是 Agent 仍可访问角色所定义的结构信息并访问其内部属性。

■ **“Resume”**：Agent 可执行自适应动作“Resume”以将所绑定角色的状态从“非活跃”调整为“活跃”状态。当 Agent 所绑定的角色处于活跃状态时，它将约束 Agent 的运行。例如 Agent 将根据角色所定义的行为来选择和执行动作。

3 基于学习和动态绑定的在线自适应决策

3.1 在线自适应决策

自适应决策是封闭自适应控制循环中的一个重要环节。在自适应系统开发过程中，通常有二种不同的自适应决策方式：离线决策和在线决策。离线决策也称静态决策，它是由开发人员在设计阶段通过显式地描述自适应策略或者规则来定义针对何种变化，对系统进行什么样的调整。目前该方法被广泛用于自适应系统的开发[5][11]。显然，这种方法对于变化可预期、适应性调整可事先规约的自适应系统而言是有效的。

然而，对于开放环境下的自适应系统而言，其开发和运行将面临以下二方面的挑战：（1）系统和环境的变化不可预知和不确定，因而开发人员无法在设计阶段给出关于适宜性需求和逻辑的清晰描述。一旦在运行时出现未预期的变化，通常做法是重新定义自适应策略或规则，随后进行重编译、加载和执行，这将导致自适应系统难以持续运行和演化[3]。（2）缺乏对策略执行效果的评判，预定义的策略一般是按照开发人员对系统以及环境的理解对在不同的状态下系统应该做的调整给出的类似“定性”的判断，并没有针对不同环境状态下执行不同的自适应操作会给系统带来的好处进行“定量”的判断。

针对上述问题，将学习引入到自适应决策过程，采用在线的方法支持系统的动态自适应决策，即使得系统自身能够在运行时通过学习的手段建立起所观察到的变化与适应性调整之间的关系，从而对自适应行为进行决策。本文的研究基于以下的基本假设：Agent 在任何时刻所绑定的角色只能有一个处于活跃状态，即不考虑 Agent 同时受多个角色约束和指导的情景。

图 3 描述了结合强化学习和动态绑定的自适应运行机制。它由三个部分组成：SA，角色和环境，其中 SA 是行为主体，环境是影响 SA 适应性的因素，角色是 SA 实施自适应的内容。SA 内部具有决策部件和行为池，SA 绑定的角色中的行为将会放入行为池中，行为的调度执行可能会对环境产生影响，从而导致环境状态发生变化。SA 将感知到的状态变化交给决策部件，决策部件结合已有的关于环境的知识，制定出是否执行或执行哪个自适应操作的决定，若选定自适应操作执行，则导致新一轮“行为执行—状态感知—在线决策”的过程。决策单元包含学习器、动作选择策略和学习到的知识。学习器根据一定的学习算法完成对 SA 在不同状态下执行的自适应操作的适应效果的学习；动作选择策略按照ε-greedy 或 Boltzmann 分布机制选择不同状态下执行的操作，从而通过为各个操作赋予一定的概率实现了“探索”和“利用”的平衡；学习到的知识通过一定的表格形式保存起来，这些表格被称为知识表。基于强化学习的自适应系统运行时将不断与环境交互，并在此过程中，对在不同状态下执行的自适应操作的好坏进行评判，形成应对不可预知、动态变化环境的知识。

3.1.1 自适应操作的执行

根据动态绑定机制，SA 可执行一系列的自适应操作以对 SA 所绑定的角色进行调整。“*join*(*r*)”的执行将使得角色 *r* 所定义的一组行为将会加载到 SA 的行为池中，SA 通过统一的行为调度器对行为进行调度执行，行为的执行作用在环境上可能会导致环境状态的变化。“*quit*(*r*)”的执行将使得行为池中对应于角色 *r* 的所有行为被删除。“*resume*(*r*)”执行后，对应于角色 *r* 的行为将被激活，行为将会重新被调度执行。“*suspend*(*r*)”执行后，对应于角色 *r* 的行为将被挂起。

3.1.2 SA与环境的交互

SA 感知到当前环境状态 s_t 以及环境反馈的评价值 r_t 后，选择执行一定的行为 b_t，b_t 的执行可能会对环境产生影响，导致环境状态从 s_t 变成 s_{t+1}，环境对 b_t 的执行效果给出评价，作为行为执行的回报值反馈给 SA，SA 感知到新的环境状态以及环境对其行为执行的反馈后，依据一定的策略再选择一个动作执行，从而触发了新一轮的交互过程。SA 在和环境的交互过程中，会将环境给出的关于自身在不同环境状态下执行的动作的评价保存起来，并

不断更新，在进行大量的交互后，将获得关于应对环境不同状态的知识。

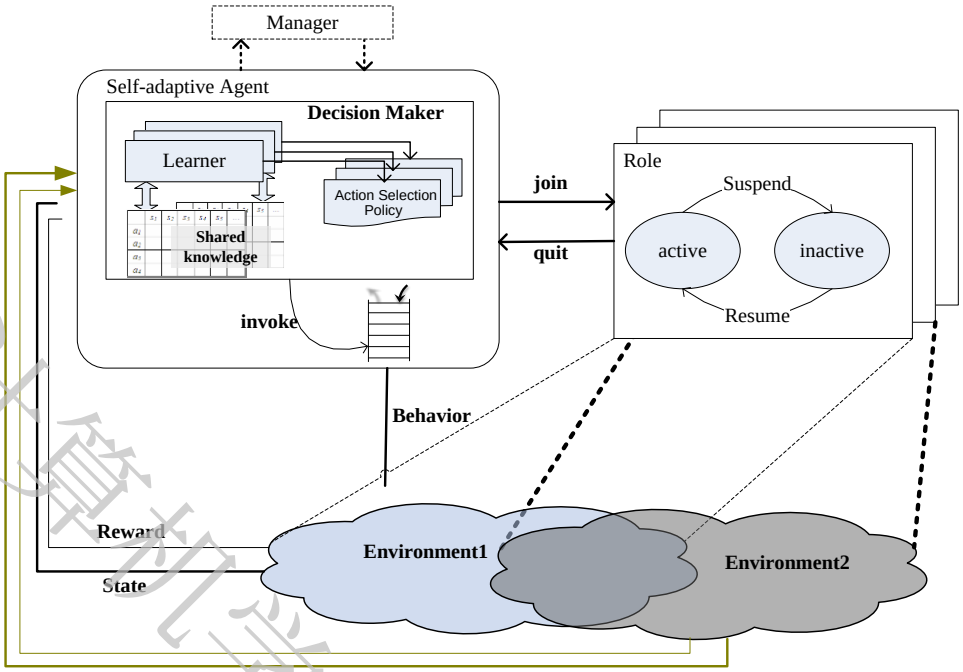

图 3　基于强化学习和动态绑定的自适应

3.1.3　SA的自适应学习过程

SA 感知到环境变化后，将环境状态（假设状态为 *s*）发送给行为选择单元；行为选择单元根据一定的选择策略，从该状态下可能执行的多个行为中选择一个最合适的（假设动作为 *b*）提交到执行单元；执行单元或直接执行行为，或通过调用一些外部接口（比如通信接口等）来执行，行为的执行可能会影响环境的状态；环境对 SA 执行行为好坏的评判通过回报来给出，评价单元获得环境给出的回报值后，将对在 *s* 状态下执行的行为 *b* 的 Q 值进行更新；对 Q 值的更新通过学习单元写进知识表中，学习单元还会根据对当前系统“探索-利用”的要求，调整学习的相关参数（比如学习率α，折扣率γ）和动作选择策略（比如ε-greedy 算法中的ε值）。SA 不断的监测环境，感知环境的当前状态，从而不断触发“选择-执行-评价-学习”过程。与此同时，适应环境的知识表也不断得到更新和维护，进而使得 SA 能够根据知识表选择出更加适应环境变化的动作，逐步改善 SA 的适应效果（如图 4 所示）。

需要强调的是，论文所提出的方法允许 SA 绑定多个角色、加载多个不同的学习器（见图 3）。SA 可能根据需要会绑定不同的角色，每个角色对应的功能和环境有所不同，针对不同的角色，SA 需要加载不同的学习器来学习角色中动作的执行，制定出能够适应环境的在线策略。

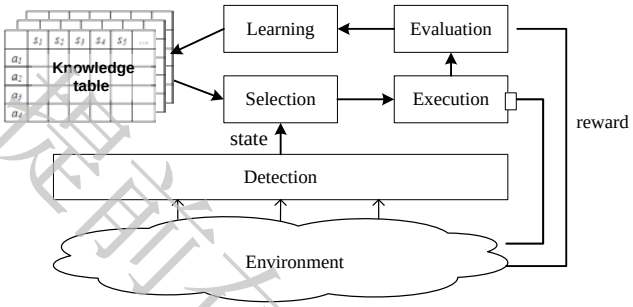

图 4　SA 的自适应学习过程

3.2　角色动态绑定的自适应学习算法

角色动态绑定的自适应性学习是指 SA 通过学习获得适应环境或自身变化的角色调整策略，即获知在何种情况下，执行何种角色调整动作会带来较好的适应性效果。其主要任务是通过与环境的持续交互，不断更新对应于不同状态下可执行的角色动态绑定动作的 Q 值，从而获得不同状态下实施适应性动作以适应环境变化的效果。角色调整动作适应性学习的算法与经典的强化学习算法有很大不同，尤其在回报值的设置和行为选择方面，这种不同是由角色动态绑定动作的特性所导致的。

- **状态**：包括环境的状态和 SA 的状态。
- **操作**：学习表中可设定的操作包括角色中的基本行为，形式如“*role.behaviour*”，其中 *role*

是指角色的名字，*behaviour* 是指 *role* 角色中包含的行为；以及角色动态绑定的四个原子动作，包括 *join(r), quit(r), suspend(r), resume(r)*，其中 *r* 是指角色。

■ **回报函数**：在角色绑定的自适应学习算法中会涉及到两类不同的适应性操作，因此回报函数也要针对这两类操作进行分别设置。其中基本行为的回报函数设置可参考强化学习中一般方式设置，通常的做法是在状态 *s* 下，如果行为 *b* 的执行有利于目标的实现，则 *r*(*s*,*b*)赋正值，否则 *r*(*s*,*b*)赋负值。而绑定角色动作回报函数的设置与基本动作的设置有所不同：（1）*join* 操作执行并不会对环境产生直接的影响，但是该动作执行完后 SA 将拥有角色所定义的行为，而这些行为的执行将会对环境产生影响。因此 *join* 动作的回报是一种延时回报，回报值的获得是通过对加入该角色后执行的一系列行为所带来的效果的综合评判后得出的。因此，设置 *join* 动作执行的回报值，可采取的一种方案是通过观察和统计加入角色后 SA 的行为执行情况来决定回报值的大小。（2）*quit* 操作的执行不会对环境产生影响，因此 *quit* 操作的回报值为 0。（3）同 *join* 操作类似，*resume* 操作的的回报也是延时回报，其回报值设置同 *join* 操作。（4）*suspend* 操作的执行情况类似于 *quit* 操作，其回报值为 0。

开放环境下 SA 的角色动态绑定自适应性学习算法如算法 1 所示，其中 *rd* 是环境返回的立即回报值；α 是学习率（$0<\alpha<1$），它控制着学习的速度；γ 为折扣因子（$0 \le \gamma < 1$）。该算法描述了假设 SA 已绑定角色 r_1 的情况下，如何通过学习绑定角色 r_2。该算法的输入是环境和 SA 的状态，输出是学习到的、对应于 r_1 和 r_2 这两个角色的知识表。该算法的核心是 *d~k* 步。角色调整动作的执行会导致 SA 的角色发生改变（如转换为角色 r_2），随之会加载不同的学习器和知识表。之后按照基本行为的学习算法对角色 r_2 行为的执行情况进行学习。学习结束后，观察或计算出按照最优方式完成角色 r_2 的任务或目标所需的总体代价（比如能量的消耗，移动的步数等），并结合绑定角色 r_2 的回报函数，给出之前在角色 r_1 下执行绑定角色 r_2 行为的回报值。之后 SA 将角色转换为 r_1，并将学习器调整到 r_1 对应的学习器，观察新的状态触发新的一轮学习过程。表 2 描述了角色 r_1 的知识表，它既刻画了 SA 执行角色的基本行为的情况，也刻画了在不同状态下执行角色调整动作的效果。通过该表，SA 可以制定出在绑定角色 r_1 的情况下，如何选择不同状态下应该执行的自适应操作的策略。当然如果该策略的基本准则是选取每个状态下的能够带来最佳执行效果的自适应操作（即 Q 值最大的操作），那么可直接将学习到的知识表映射成自适应策略（$\pi: S \to A$），该策略可表示为：

$$\pi(s_i)=a_k, a_k \in \left\{ a \middle| Q(s_i,a)=\max_j Q(s_i,a_j) \right\}$$

该策略选取每一个状态 s_i 下的 Q 值最大的操作。如果 Q 值最大的操作有多个，则任选一个。

表 2 角色 r_1 的 Q 表

	s_1	s_2	…	s_n
$r_1.b_1$	$Q(s_1, r_1.b_1)$	$Q(s_2, r_1.b_1)$	…	$Q(s_n, r_1.b_1)$
$r_1.b_2$	$Q(s_1, r_1.b_2)$	$Q(s_2, r_1.b_2)$	…	$Q(s_n, r_1.b_2)$
…	…	…	…	…
$r_1.b_m$	$Q(s_1, r_1.b_m)$	$Q(s_2, r_1.b_m)$	…	$Q(s_n, r_1.b_m)$
$join(r_2)$	$Q(s_1, join(r_2))$	$Q(s_2, join(r_2))$	…	$Q(s_n, join(r_2))$

算法 1: 角色绑定动作的学习算法 *RoleBindingActionLearning*
假设:　自适应 Agent 已经绑定角色 r_1
输入:　自适应 Agent 及其驻留环境的状态 s
输出:　$QValueTable(r_1)$, $QValueTable(r_2)$
{
1. FOR 每一个(s, a), 初始化表 $Q(s, a)$, 其中 $Q(s, a) \in$ QValueTable (r_1);
2. 观察当前状态 s;
3. 一直循环直到 s 是终止状态:
   a) 选择动作 a;
      IF (a 是一个行为) THEN do (b ~ c, i ~ k) ELSE do (d ~ k)
   b) 根据策略(如ε-greedy)执行行为 a;
   c) 获得奖励值 rd;
   d) 执行 a (如 $join(r_2)$), 改变自适应 Agent 所绑定的角色;
   e) 加载和初始化针对角色 $r2$ 的学习器;
   f) 根据基本行为开始学习，结果保存在表 QValueTable(r_2)中;
   g) 观察和计算实现角色 $r2$ 的任务所需的成本，根据绑定角色 $r2$ 的奖励函数，获得绑定动作(rd)的奖励;
   h) 卸载针对角色 $r2$ 的学习器,重新加载针对 $r1$ 的学习器,继续学习;
   i) 观察新的状态 s';
   j) 采用如下公式更新 $Q(s, a)$:
      $$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[rd + \gamma \max_{a'} Q(s',a')]$$
   k) 将状态 s 赋为 s'.
4. 返回 QValueTable(r_1), QValueTable(r_2)
}

4　自适应软件的构造和运行技术

本文更加关注的是如何借助于动态绑定的自适应机制和增强学习方法来支持开放环境下自适应系统的开发。本节将从软件模型、开发框架和方法、支撑环境三个方面介绍自适应软件的构造和运行技术。

4.1　自适应软件模型

通过对软件 Agent 的反应式模型进行扩展，提出了如图 5 所示的基于增强学习的 SA 软件模型。它包含感知、决策、行为池以及效应器四个主要部件，决策部件包括学习器、知识表和动作选择策略。感知部件负责感知环境变化，感知结果以环境状态的形式反馈到决策单元。感知单元也可作为环境对 SA 行为评价的转发器，将环境给予 SA 的回报值转发到学习单元。决策单元的学习器以环境的状态和环境给予动作的回报值为输入，利用学习算法学习在不同的环境状态下 SA 执行的自适应操作的适应效果，并将这种学习结果以知识表的形式保存。在选择当前状态下可执行的自适应操作时，SA 可根据动作选择策略（比如ε贪婪策略）从知识表中当前状态下的多个行为中选择出合适的行为或行为序列，并将选择的行为放到 SA 的行为池（Behaviour Pool）中。行为池中的行为会通过 SA 的行为调度器统一调度执行，行为的执行可能会影响环境。效应器的功能是将行为的执行效果施加到环境上，比如通过事件发布器向外发布 SA 状态的变化或动作的执行，从而使得关注该 SA 的其他 SA 可以感知到这样的变化。或者通过调用外部接口，对物理环境施加影响。通过效应器，SA 可以将自身行为的执行施加到对环境的影响上。环境状态变化会被感知器捕获，从而又会触发新一轮学习和适应过程。

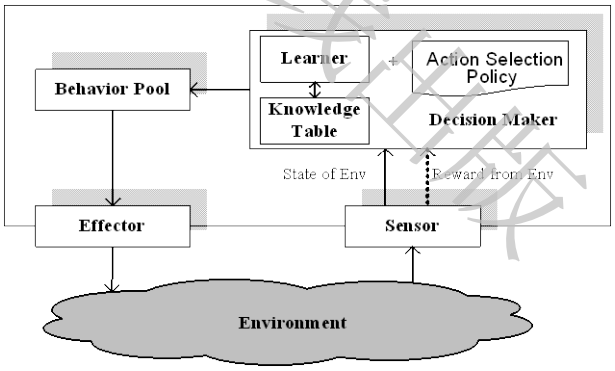

图 5　自适应 Agent 的软件模型

4.2　自适应软件构造框架

基于动态绑定自适应机制和强化学习方法的自适应系统构造框架如图 6 所示。开放环境下自适

应系统的开发涉及以下几个关键部件的设计和实现：自适应 Agent、角色、学习器和环境。为了更好地支持自适应系统的开发，我们提供了一组可重用构件来封装上述基本功能，包括 SA 类封装了自适应 Agent 的动态绑定角色的基本操作（如 Join、Quit 等）、Role 基类封装了角色的基本功能、环境基类 Environment 和学习器基类 Learner 等等。基于动态绑定自适应机制和强化学习方法的自适应系统构造框架如图 6 所示。

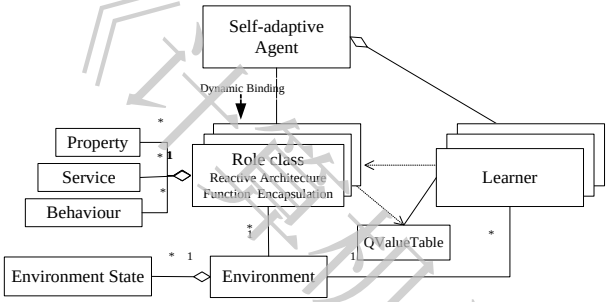

图 6 基于学习和动态绑定机制的自适应系统开发框架

开发人员可以通过继承和重用 Role 基类来定义与应用相关的角色类。角色类实现了在组织上下文中一组 Agent 的共同业务逻辑，封装了与业务逻辑相关的属性、行为、服务和环境上下文。其中，属性定义了角色的性质和资源，服务通过行为实现，环境上下文是对角色所在环境的描述。角色中的行为是一个反应式的规则，它定义了当特定事件发生、某些条件得到满足时需执行的一组动作序列。

学习器是系统的学习单元，其学习功能的实现依赖于角色中定义的行为。开发者可以继承学习器抽象类来得到不同类型的学习器类。一个 SA 根据不同的学习目标可以加载不同的学习器实例。SA 和环境的交互过程中，通过试错的手段，执行一定的行为对环境施加影响，并将环境的状态和行为执行后环境的反馈输入到加载的学习器中，学习器根据一定的学习算法，学习适应环境的知识，并将这些知识存储在知识表（如 QValueTable）中。学习器中的知识随着 SA 和环境的不断交互而不断更新。SA 执行角色功能时可参考知识表中在不同状态下执行不同行为的效果来选择最为合适的行为执行，角色和知识表之间是依赖关系。

4.3 支撑软件环境SADE+

通过对多Agent系统开发框架JADE[10]进行扩展，开发了开放环境下自适应系统的开发和运行支撑平台 SADE+，其整体框架可分为三个层次：基础层、运行层和开发层（见图 7）。

- 基础层借助 JADE 的基础设施实现了不同节点、不同平台 Agent 之间的通信，为分布式多 Agent 系统的交互提供了支持，并在 JADE 提供的基于消息通信的基础上，实现了事件服务，从而为环境感知提供了基础。扩充了 Agent 生命周期管理功能，实现了对 SA 生命周期的管理。

- 开发层为自适应软件的开发提供支持。自适应策略描述语言提供了 SADL 的语言规范[11]；自适应软件开发包为 SA、Role、自适应策略以及学习器的开发提供了基本的编程规范和基类支持；SADL 编辑器和编译器为开发人员制定自适应策略提供辅助功能；学习算法库提供针对不同应用的学习算法。

- 运行层提供了支持 SA 运行的功能部件。其中自适应操作为 SA 提供了基本的自适应能力；通过加载和使用学习器，SA 具有了在线学习适应环境的知识的能力；软件监控器负责对 SA 的自适应操作的执行情况进行监控；自适应策略的加载和升级实现了策略的静态修改、动态升级；冲突检查器为 SA 的正常运行提供保障。

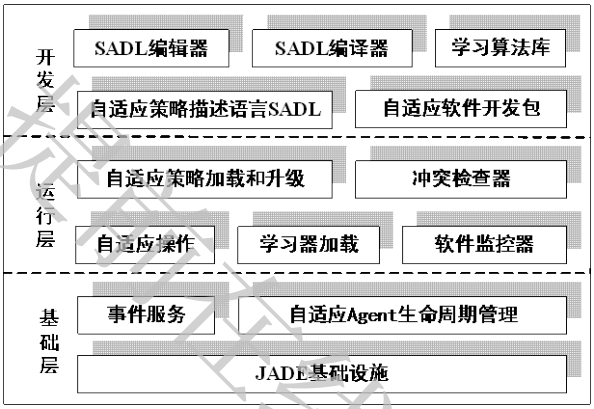

图 7 自适应系统的开发和运行支撑环境 SADE+

4.4 开发方法和案例分析

本节通过具体的案例分析来介绍开放环境下自适应系统的开发过程。该案例模拟了开放环境下猎狗追击猎物的组织场景。组织中有二类 Agent：机器狗和猎物，它们运行在 N×N 的格子环境中，格子环境具有多样化的地形，包括沙地、泥泞地等，并且是开放的，即随着时间的流逝，这些地形会消逝或改变，比如平地可能因为雨水的冲刷而变成泥泞地，障碍物也会随机出现或消失，因而无法事先完全预知其变化。运行于网格中的猎物会按照不同的方式在网格中移动，比如随机移动、围绕网格做顺时针移动，或在某条路径上循环移动。开始时机

器狗会被随机放置在格子中的某个位置，之后不断移动来寻找猎物，机器狗每移动一次会消耗一定的能量，且在不同的地形上移动或碰到障碍物消耗的能量有所不同。当机器狗的能量不足时，要及时前往供给站补充能量，而后再次追击猎物。为此，针对机器狗要完成的两种不同的功能，我们设计了两类角色：追击者角色（Catcher）和补给者角色（Supplier）。当机器狗扮演追击者角色时，主要任务是尽快的追到移动的兔子；当追击过程中机器狗自身所剩能量不足时，机器狗将扮演补给者角色，寻找能耗最小的路径，以最节能的方式移动到补给站来进行能量补充。为简化案例分析，我们有以下假设：（1）机器狗和猎物运行在 10×10 的网格中；（2）环境的变化（如障碍物）自动产生并且不可预测；（3）在任何时刻 Agent 可执行以下四个方面的基本动作{moving up, moving down, moving left, moving right}；（4）机器狗可以感知环境中的任何变化以及猎物实施的任何动作。

在案例的实际运行中，何时由追击者角色转换成补给者角色，是机器狗能够尽可能多地捕获到猎物的关键。一方面，机器狗必须保证能够实现能量补给；另一方面，还要保证转换角色前能量尽可能多的用到追击猎物上。否则，要么可能造成机器狗角色转换不及时，导致不能到达供给站；要么在剩余大量能量时转换到补给站，导致了不必要的角色转换，既影响了捕获猎物，又增加了不必要的角色转换开销。

下面两个场景分别展示了自适应系统如何通过学习在动态不确定环境下在不同层面下的自适应：

（1）机器狗扮演追击者角色的过程中，机器狗所处的环境（格子世界的障碍物和地形的位置等）是开放的，会动态地发生变化，并且这种变化往往具有不可预知和不确定等特点。例如，地面会由于下雨而导致泥泞和打滑，从而导致更多的能量损耗；雨水的冲刷可能会产生新的水道，导致机器狗无法通过，因而需要绕道并提前去补充能量等等。追击的猎物也在不断的改变位置，机器狗此时的目的就是在这种动态的环境下通过学习，得到行为的最优策略，移动最少的格子追击到猎物。

（2）在机器狗扮演追击者角色过程中，机器狗会不断监测自身的能量状态，当能量处于某个临界值时，需要到补给站进行能量补给，此时机器狗会执行“suspend”适应性元操作，将“追击者”角色置为“非活跃”状态，并通过执行“join”适应性元操作，扮演和绑定“补给者”角色，从而来补充能量。一旦机器狗补充了足够的能量，它将执行“quit”适应性操作以退出“补给者”角色，并通过执行“resume”适应性操作将“追击者”角色置为“活跃”状态，重新追击猎物。

由于格子世界中地形和障碍物的位置不断变化，因此机器狗在不同时刻处于不同位置时，这个临界值也是不一样的。这个临界值在机器狗扮演补给者角色时通过学习得到，其值为到补给站所消耗的最小能量。

针对案例特征，我们按照图 8 所示的自适应系统的开发过程进行设计和实现。首先，识别和设计组织中的角色，开发者需要根据需求识别组织中的角色，分析和定义角色的属性、服务和行为，并通过继承基类 Role 对角色进行设计和实现。在该案例中，可以识别出“Catcher”、“Preyer”、“Supplier”三类角色。其中，“Catcher”角色封装了作为追逐者的一组属性（如位置、能量等等）、环境（即开放的网格）和行为（如“moving”, “sensing a prey agent in the environment”等等）。

其次，设计和实现系统中的 Agent。开发者需要识别、构造和实例化自适应系统中的软件 Agent。如果该 Agent 具有自适应行为，那么它需要继承基类 SAgent。该案例有二类不同的 Agent：HunterAgent 和 PreyAgent。其中，HunterAgent 是一个自适应 Agent，它能够根据环境及其自身的变化，不断地调整它所扮演的角色，以持续性地获得能量并追逐猎物。因此，HunterAgent 应继承基类 SAgent。

第三，设计和实现学习器。开发者应为自适应 Agent 设计和实现学习器以支持其在线的自适应决策。该案例可以识别出二类学习器：“CatchLearner”和“SupplyLearner”。“CatchLearner”负责学习有关如何扑捉猎物的知识，“SupplyLearner”负责学习如何消耗最少的能源到供应站获取能源。学习器的设计一般涉及以下四个子步骤：

（1）识别和分析 SA 感兴趣的环境和自身状态。环境的信息多种多样，SA 内部可能也有很多的属性。开发者应从中选择那些与 SA 自适应相关的环境和自身状态信息，以支持 SA 开展学习。案例中“CatchLearner”学习器感兴趣的状态包括：追逐者和猎物的位置、猎物的移动方向、网格环境中的障碍物等等。

（2）选择操作集。学习的目的是通过学习建

立环境状态与动作的映射（即策略），从而使 Agent 的动作执行能够从环境中获得最大的回报。

该学习器的操作有{Catcher.moveUp(), Catcher.moveDown(),Catcher.moveLeft(),Catcher.moveRight(),Catcher.catchRabbit(),join(Supplier),quit(Supplier)}。

该操作集中既包含了角色 Catcher 的内部行为，比如向上、向下、向左、向右移动等，也包含了 SA 可执行的角色转换操作。

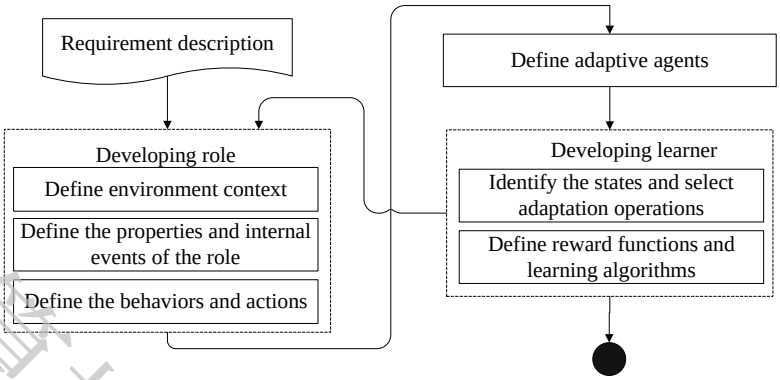

图 8 开放环境下自适应多 Agent 系统的软件开发过程

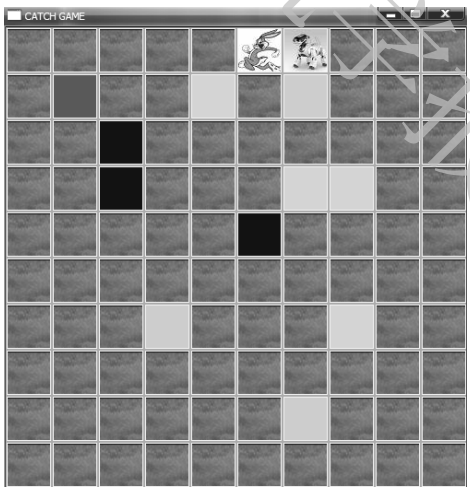

图 9 案例运行截图

（3）定义回报函数。开发者需要为每一个学习器的操作定义回报函数。在本案例中，如果 HunterAgent 碰到障碍物，那么其回报为负数；如果 HunterAgent 的剩余能量较多（如大于 50）或者剩余的能量无法支持其移动到供应站，那么其回报为负数，否则为正数，具体设计如下公式。需要强调的是，本案例分析中的一些数字是经验值，可以根据软件运行的情况对其进行调整。

$$reward=\begin{cases}\dfrac{50}{energy_{remain}}, \text{if } energy_{remain}\le 50.\\ -100, \text{if } \begin{pmatrix} energy_{remain}>50\\ \text{or has not moved to supply station}\end{pmatrix}.\end{cases}$$

（4）设计学习算法。该案例的学习算法见 3.2 节所示，SADE+已经将该学习算法封装在 Learner 基类中，开发者需在定义各个学习器时继承 Learner。

表 2 的角色调整动作适应性学习过程反映了知识融合情况，即在 SA 角色转换到 r2，执行 r2 的行为并完成其功能后，观察并计算执行 r2 中行为的总体情况，从而结合绑定角色 r2 的回报函数，给出之前执行绑定角色 r2 操作的回报值，进而更新绑定角色 r2 对应的 Q 值。图 10 以知识表转换和更新的形式对上述知识融合过程进行了剖析。SA 的学习从角色 r1 对应的知识表开始，中间转换到角色 r2 对应的知识表，之后又结合 r2 知识表来更新 r1 表中的相关内容。上述过程可理解为，角色行为的执行效果可作为角色转换条件的依据。

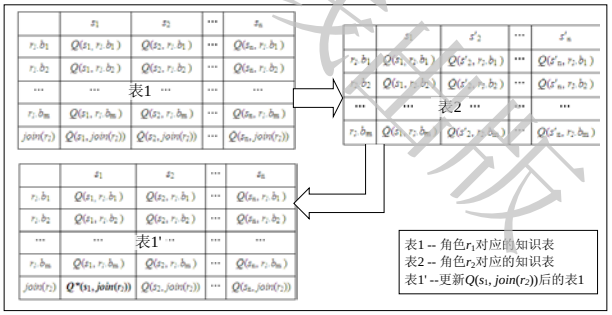

图 10 知识融合示图

我们在 SADE+平台上成功开发了上述案例，图 9 是该案例的运行示意图，图中方格为平地，其他不同颜色的方格表示环境中多样化的障碍物，如沙地、泥泞地等等。

下面通过两个情景对机器狗的执行效果进行实验，通过对实验结果的比较，可看出 SA 在具有学习功能后，能够适应动态、不确定环境，并更好的完成任务。

情景一：在机器狗能量充足情况下，完成单角色任务情况

为了测试通过学习机器狗能够更好的适应环境变化，完成追击者角色捕获猎物的任务，我们让机器狗在能量充足的情况下按照两种不同的方式追击猎物：一种是随机移动的方式，另一种是机器狗带有学习的移动。本实验是在机器狗拥有 5000 个单元能量时，比较通过两种不同的捕获猎物方式完成捕获猎物任务的效果。其中，在学习的方式中，采用 3.2 节中的自适应学习算法，取参数γ=0.9，学习过程按照ε=0.8 的ε-greedy 策略选择动作。

图 11 中上下两条曲线分别展示的是按照随机移动的方式和带有学习的方式捕获猎物的情形。横坐标表示的是捕获猎物的次数，纵坐标表示的是每两次捕获猎物需要移动的步数的平均值。该图表明，尽管存在极个别的情况（如横坐标为 16 时基于学习捕获猎物所花费的步数大于随机移动的步数），但是从整体上看，经过一段时间的学习，机器狗会用小于随机移动的步数捕获到猎物，所花费的步数浮动也远小于随机移动的程度，这说明本文提出的自适应学习的方法整体上提高了 Agent 适应环境的能力、降低了损耗。导致极个别异常情况的原因是多方面的，由于环境的开放性、Agent 对环境感知和自身信息的有限性，Agent 对适应性行为的学习效果必然存在一个修正和提高的过程。

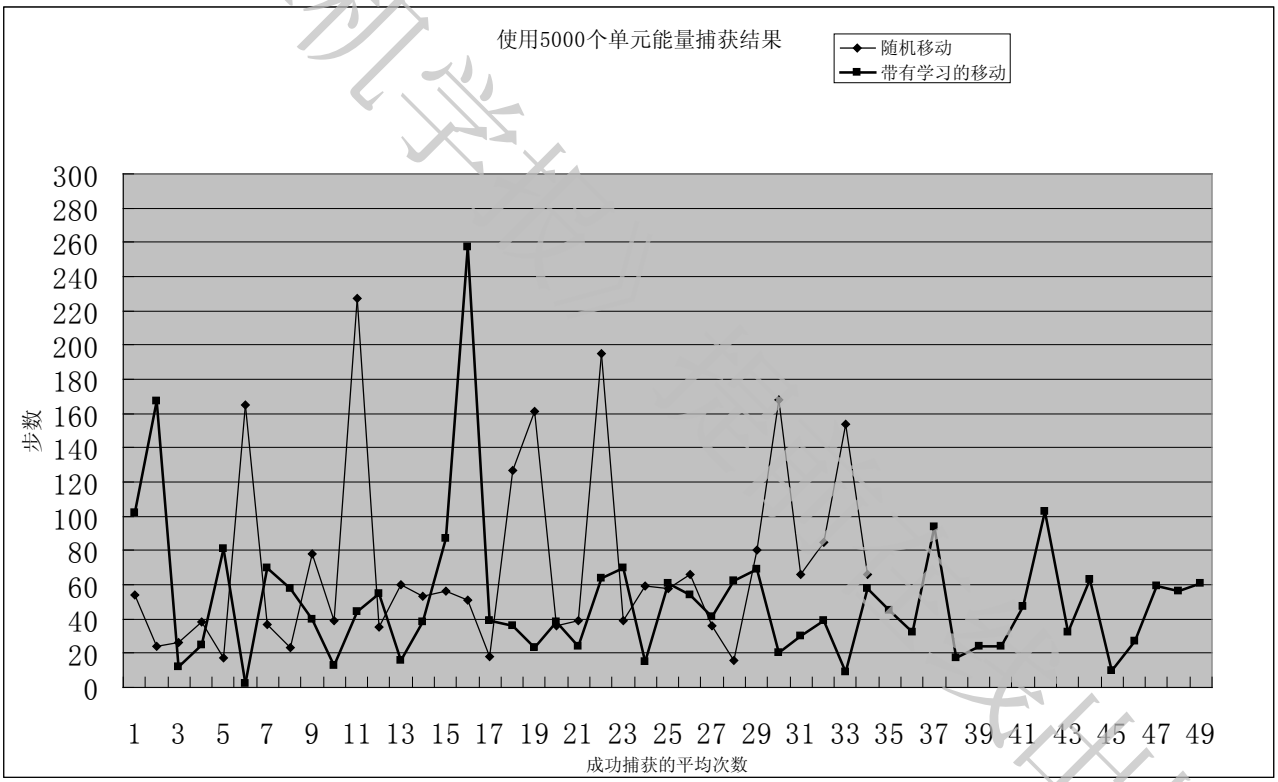

图 11　能量充足条件下机器狗捕获猎物情况

情景二：在机器狗能量受限情况下，保证尽可能多的捕获猎物，并及时补充能量

在这种情景下，要求机器狗能够根据自身能量的情况自动转换角色，从而及时补充能量，进而可以继续下一次的猎物捕获过程。针对此情景，机器狗将按照两种不同的方式捕获猎物。

方式一：机器狗基于学习的方式移动捕获猎物，并且在能量小于某值（设为 40 个单元的能量）时转换角色为补给者角色，并基于学习的方式移动至补给站，开始补给能量。方式二：机器狗基于学习的方式移动捕获猎物，并且在能量小于某值（初始设为 60 个单元的能量值，之后通过学习不断调整该值）时转换角色为补给者角色，并基于学习的方式移动至补给站，开始补给能量。

在本实验中，设定机器狗初始能量为 200 个单元；在学习的方式中，采用 3.2 节中的学习算法，取参数γ=0.9，学习过程按照ε=0.8 的ε-greedy 策略选择动作。猎物按照顺时针方向移动。

对于这两种方式，我们称这样的过程为一次执行结束。即在机器狗捕获猎物过程中，如果能量不足则进行角色转换，角色转换后如果不能够成功补给能量，说明机器狗已耗尽能量且未到达补给站，一次执行结束。或者能够成功补给能量，则再进行一轮猎物捕获活动，直到能量耗尽为止，一次执行结束。在本实验中，我们对这两种方式的每一次执行进行一次统计，统计补给能量是否成功，以及在这次执行中捕获的猎物数。然后对每 100 次执行进行一次比较，并称其为一轮。下面我们给出第五、十、十五轮执行的效果比较。

图 12 给出机器狗按照两种方式进行角色转换后，成功补给能量的次数比较。可以看出随着学习的进行，按照学习的方式不断更新角色转换条件，成功转换角色并补给能量的次数越来越多，且要好于在固定条件下转换角色的情况。

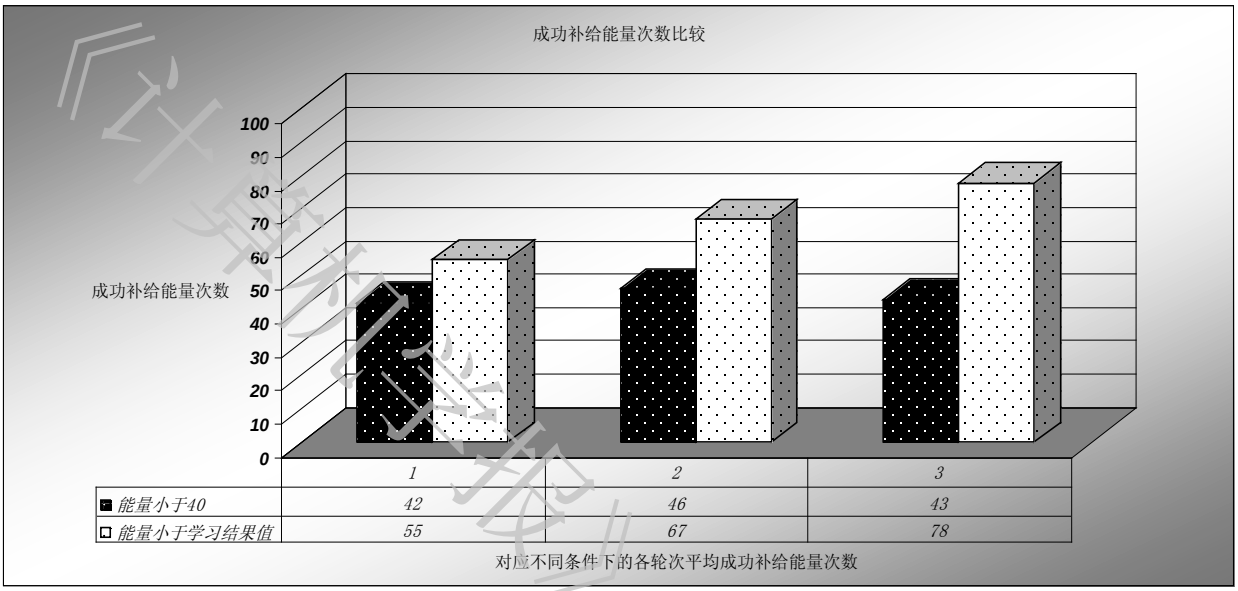

图 12 按照不同的方式角色转换并补给能量成功次数比较

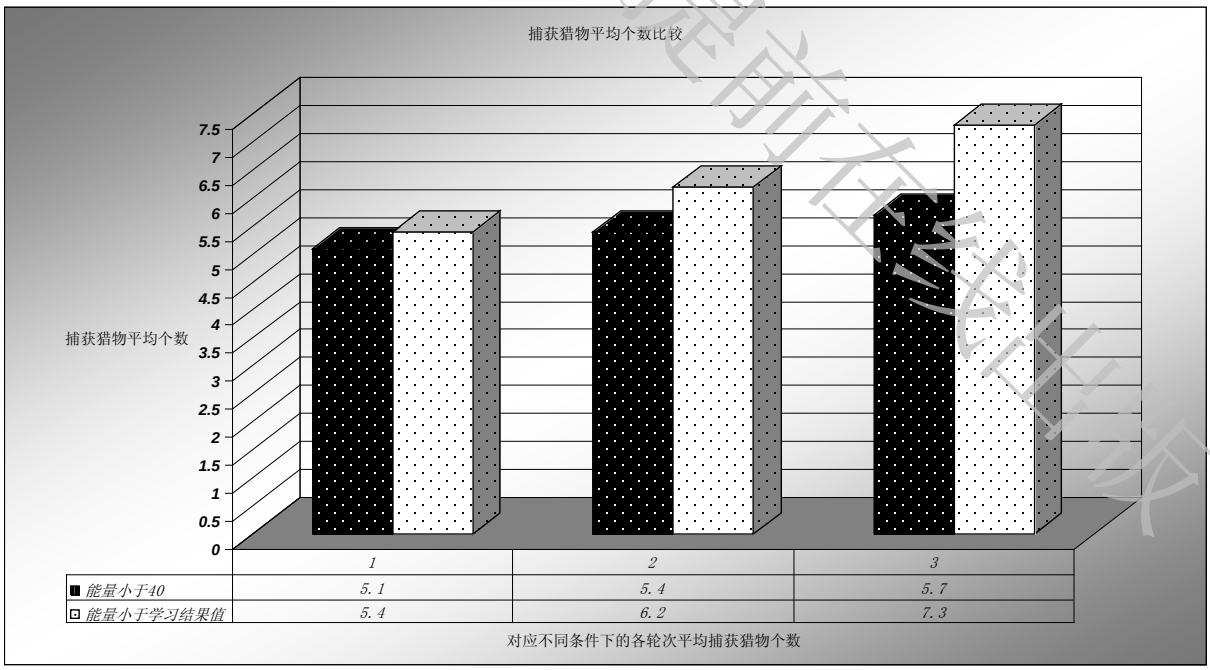

图 13 按照不同的方式捕获猎物平均个数比较

图 13 给出了在不同的角色转换方式下，机器狗每一轮捕获猎物的平均个数比较。从实验的结果可以看出，通过学习并进行自适应调整，处于动态环境中的机器狗在保证了最多捕获猎物的同时，又

能及时完成能量补给，更好的完成了捕获猎物的任务。

需要说明的是，本案例中 SA 感兴趣的环境状态在系统设计时已经确定。若要适应系统运行期间环境中出现的新的环境变化，如产生了新的物质或障碍物，则需要一定的人为干预（如图 3 中的 Manager 与 SA 的交互），比如系统开发人员或管理人员要识别这种新物质，并获取 SA 与该物质发生碰撞后给整个系统带来的影响（即环境回报值），从而设计回报函数，并将新的环境状态加入到知识表中，而后 SA 可在新的知识表和回报函数基础上，重新学习适应环境变化的策略。

5　相关工作分析

由于系统部署环境（如 Internet）的变化以及对系统灵活性和健壮性等方面提出更高的要求，有关适应性系统的研究近年来引起了学术界和工业界的高度关注和重视，如开放、动态、难控网络环境下的网构软件[1]、软件密集型的超大规模系统[2]等等。尽管自适应系统的研究已经有较长的时间并在许多工程领域取得了诸多成功，但是如何采用系统、高效的方式来支持自适应软件系统的工程化开发仍然是软件工程面临的一项重要挑战，包括需求、建模、实现、验证等等[3][13]。尤其是，自适应软件系统的研究是多学科交叉的，包括软件工程、控制论、决策论、人工智能、网络和分布计算等等[4]。

在自适应系统软件工程领域，目前人们大多借助于软件体系结构技术、基于构件软件工程、面向方面程序设计、中间件、计算反射[7]等方法来支持自适应软件系统的工程化开发。体系结构模型及其语言（如体系结构描述 ADL）可用于自适应系统设计态的建模以及运行态的管理，人们已经提出了许多基于图形（如 Bi-Graph）、代数等动态体系结构描述语言，该方面的代表性工作是 Garlan 的 Rainbow[5]以及[9][14]。基于构件的技术通过构件模型来设计和实现自适应软件，并通过可重用的自适应引擎对构件的自适应进行管理[4]。它通常借助于构件框架（如 COM/DCOM, .NET, Enterprise Java Beans, COBRA Component Mode），采用接口协议、运行时绑定等技术手段，允许开发者在运行时通过增加、删除、重配构件以实现系统的适应性[7]。面向方面的编程（AOP）支持开发者用方面来封装自适应逻辑和关注点，并通过动态织入等机制实现软件系统运行时的自适应。上述研究工作均基于自适应软件系统的以下开发和运行技术框架，即开发者在设计阶段显式给出自适应逻辑和决策的描述，软件系统在运行阶段根据所观察到的环境或自身变化，通过对预定义自适应逻辑和决策的解释和分析来实施自适应。

软件密集型系统的适应性可以在不同层次上进行，如控制数据[15]、构件和体系结构[5]等。近年来，借助于软件 Agent 技术来开展自适应系统的软件工程研究成为这一领域的重要发展趋势，这是因为人们认识到软件 Agent 可以为自适应软件系统的开发和运行提供一系列的技术支持，如自主决策、环境感知、自然建模等等。现有该方面的研究大多集中于借助 Agent 抽象来支持对自适应软件系统进行建模、分析和设计，包括元模型[6]、建模语言、开发方法学（如 Gaia[6][16]、O-MaSE[16][17]、ADELFE[6]、MetaSelf[18]）。其中许多研究借助于社会组织的思想来抽象自适应系统、分析自适应的机制和方法，如通过角色的动态重分配（Role Dynamic Reallocation）[19]或者动态角色扮演（Dynamic Role Playing）[20]来对组织的适应性进行分析和设计。显然，如何将这些高层的自适应机制和方法应用于实现阶段以支持自适应系统的构造和运行是该领域研究面临的挑战之一。

在自适应软件的构造和实现方面，目前有以下几种不同的方法。第一，提供程序设计语言来支持自适应系统的编程，如面向 Agent 的程序设计语言 3APL[21]和 SLABSp[22]。3APL 支持对角色的四种操作如 enact、deact、activate 和 deactivate，能够支持开发具有动态调整角色能力的 Agent。但是它们对 Agent 扮演角色进行了一些限制，如同一时刻 Agent 扮演的角色中只能有一个处于活跃状态，其它的角色必须处于非活跃状态。SLABSp 基于 Caste 抽象和机制，每个 Caste 封装了环境、属性和行为，它运行程序员在行为中显示地定义 Agent 如何根据环境的变化加入（Join）一个 Caste 或者退出（Quit）一个 caste。第二，提供独立的自适应策略或者规则描述语言来对自适应逻辑进行显式描述，并与其它语言所定义的业务逻辑相互融合，如自适应规则描述语言[12]用于基于构件的自适应系统开发、自适应策略描述语言 SADL 用于定义自适应逻辑[11]。第三，软件开发框架，如 ROAD(Role-Oriented Adaptive Design)[23]。这些方法和语言均要求开发

人员在开发阶段显式地定义和描述系统的自适应逻辑。为了应对不可预测的变化和自适应需求，[4]强调需要将学习方法引入自适应系统的研究，然而现有的许多研究[24][25]往往关注于基于学习的自适应机制和算法，忽视了与软件工程的思想和方法相结合，从而为自适应系统的开发提供系统的工程化方法支持。

6 结论和进一步工作

由于环境变化的不确定性和不可预测性，软件开发人员难以在设计阶段准确地预期各种变化并借助于自适应策略描述语言或者程序设计语言来清晰地定义自适应逻辑，因而开放环境下自适应系统的开发是当前自适应系统软件工程面临的一项重大挑战。为了解决这一挑战，自适应系统的软件工程技术需要实现二方面转变，即将原先由开发人员在设计阶段给出的自适应逻辑延迟到由系统自身在运行阶段来完成，自适应决策需要由离线方式转化为在线方式。

本文提出了基于角色动态绑定和增强学习的自适应软件技术。该方法基于 Agent 和组织抽象的自适应系统元模型，提出了角色动态绑定的自适应机制，借助于增强学习方法，使得自适应软件 Agent 能够通过对非预期变化及其适应性行为的学习，实现在线的自适应决策。为了支持开放环境下自适应系统的工程化开发和运行，论文设计了基于角色动态绑定的自适应学习算法以及封装该算法的自适应软件 Agent 模型，建立了其软件开发框架，设计并实现了相应的支撑环境 SADE+。除了本文所介绍的案例之外，我们还针对自适应网上交易系统、自适应机器人等进行了应用案例的开发和分析，以分析和展示论文工作和支撑平台有效性等等，由于篇幅限制在此不一一作介绍。本文的工作建立在动态绑定机制的形式语义基础之上[10]，不同于先前工作[11]，本文为开放环境下自适应系统的开发和运行提供了系统的方法和环境支持。

进一步的研究包括：（1）本文的研究主要针对构件层面上的自适应，实际上 MAS 组织的自适应可以在不同的层次上进行，包括：属性层、个体层、组织结构层、组织间层，而且不同层次上的自适应往往具有相关性。因此需要研究如何基于社会组织思想并提供统一的框架来规约和设计不同层次的自适应机制，并提供工程化的技术支持。（2）本文的研究主要针对非预期变化的在线自适应，但是对于许多自适应系统而言，离线和在线的自适应可能都需要，因此需要研究如何实现二者之间的融合来支持多样化自适应系统的工程化开发。（3）本文的研究基于以下基本假设：Agent 在任何时候所绑定的角色至多只有一个处于活跃状态。后续我们将研究多角色绑定情况的自适应性学习问题。（4）本文研究的重点侧重于自适应系统的适应性机制及其实现方法，对于自适应系统而言环境无疑是非常重要的，本文对开放环境的研究集中于环境的变化及模型以及如何根据对环境状态的学习来实现系统的自适应，至于开放环境的其他特征（如边界的不确定和要素的变化）以及相应的问题（如对多样化环境的有效感知、抽象和表示等）将是我们下一步的研究重点。

参 考 文 献

[1] Lv Jiang, Xiaoxin Ma, Xianping Tao, Feng Xu, Hao Wu, Research and Development of Internetware. Science in China Series. E Information Sciences, 2006, 36(10):1037-1080.

[2] Linda Northrop, Ultra-Large-Scale Systems: The Software Challenge of the Future, Software Engineering Institute, Carnegie Mellon University, Software Engineering Institute, 2006.

[3] Cheng, B., et al.. Software Engineering for Self-Adaptive Systems: A Research Roadmap//Proceedings Of Software Engineering for Self-Adaptive Systems, Heidelberg, Germany, LNCS 5525, 2009: 1-26.

[4] Mazeiar Salehie and Ladan Tahvildari, Self-adaptive software: landscape and research challenges. ACM Transactions on Autonomous and Adaptive Systems, 2009, 4(2):1-42.

[5] Garlan, D., Cheng, S.W., Huang, A.C. Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer, 2004, 37(10): 46-54.

[6] Juan, T., Sterling, L. A Meta-model for Intelligent Adaptive Multi-Agent Systems in Open Environments//Proceedings of Autonomous Agent and Multi-Agent System, Melbourne, Australia, ACM Press, 2003: 14-18.

[7] Carole Bernon, Marie-Pierre Gleizes, Sylvain Peyruqueou, Gauthier Picard, ADELFE: a Methodology for Adaptive Multi-Agent Systems Engineering//Proceedings Of Engineering Societies in the Agents World III, Berlin, Germany, LNCS 2577, 2003:70-81.

[8] Mckinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B., Composing Adaptive Software. Computer, 2004, 37(7): 56-64.

[9] Yoder, J.W., Balaguer F., Johnson, R. Architecture and design of adaptive object models, ACM SIGPLAN Notices, 2001, 36(12): 50-60.

[10] Mao, X.J., Shan, L.J., Zhu, H., Wang, J. The Adaptive Casteship Mechanism for Developing Multi-Agent Systems, International Journal of Computer Applications in Technology, 2008, 31(1/2): 17-34.

[11] Menggao Dong, Xinjun Mao, Running Mechanism and Strategy Description Language SADL of self-adaptive Multi-Agent System, Journal of Software, 2011, 22(4): 609-624.
(董孟高，毛新军，自适应多 Agent 系统的运行机制和策略描述语言 SADL，软件学报，2011, 22(4), 609-624.)

[12] Qianxiang Wang. Towards a Rule Model for Self-adaptive Software, ACM SIGSOFF Software Engineering Notes. 2005, 30(1): 1-5.

[13] Thomas Kuhne, Current Trends for Self-* Systems - Research Roadmap for Self-Adaptive Systems, Seminar SS Report, 2011.

[14] Lee, S., Oh, J., Lee, E. An Architecture for Multi-agent Based Self-adaptive System in Mobile Environment//Proceedings Of Intelligent Data Engineering and Automated Learning, Brisbane, Australia, LNCS 3578, 2005:494-500.

[15] Roberto Bruni, Andrea Corradini, Fabio Gadducci, A Conceptual Framework for Adaptation//Proceedings of 15th the International Conference on Fundamental Aspects of Software Engineering, Tallinn, Estonia, Springer, 2012:240-254.

[16] Cernuzzi, L., Zambonelli, F., Dealing with Adaptive Multi-Agent Organizations in the Gaia Methodology//Proceedings of International Workshop on Agent-Oriented Software Engineering, Hakodate, Japan, LNCS 3950, 2006: 109-123.

[17] Scott A. DeLoach, O-MaSE: A Customizable Approach to Designing and Building Complex, Adaptive Multiagent Systems, International Journal of Agent-Oriented Software Engineering, 2010, 4(3): 244-280.

[18] Giovanna Di Marzo Serugendo, John Fitzgerald, Alexander Romanovsky, and Nicolas Guelfi, MetaSelf - A Framework for Designing and Controlling Self-Adaptive and Self-Organising Systems, School of Computer Science and Information Systems, Birkbeck College, London, UK, Technical Report BBKCS-08-08, 2008.

[19] Mark Hoogendoorn and Jan Treur, An Adaptive Multi-Agent Organization Model Based on Dynamic Role Allocation, International Journal of Knowledge-Based and Intelligent Engineering Systems, 2009, 13(3/4): 119-139.

[20] Hilaire, V., Koukam, A., Gruer, P. A Mechanism for Dynamic Role Playing//Proceedings of International Workshop on Agent Technology, , Melbourne, Australia, LNAI 2592, 2003: 36-48.

[21] Dastani, M., Birna van Riemsdijk, M., Hulstijn. Enacting and Deacting Roles in Agent Programming//Proceedings of International Workshop of Agent-Oriented Software Engineering, Utrecht, The Netherlands, LNCS 3382, 2005: 189-204.

[22] Ji Wang, Rui Shen, and Hong Zhu. Towards agent oriented programming language with caste and scenario mechanisms//Proceedings Of International Joint Conference on Autonomous Agents and Multiagent Systems, Utrecht, Netherlands, ACM Press, 2005:1297-1298.

[23] A. Colman and Han. Roles, Players and Adaptive Organisations. Applied Ontology: An Interdisciplinary Journal of Ontological Analysis and Conceptual Modeling, IOS Press, 2007, 2(2):105-126.

[24] Kim, D., Park, S. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software//Proceedings Of International Conference on Software Engineering for Adaptive and Self-Managing Systems, Vancouver, BC, Canada. 2009: 76-85.

[25] Amoui, M., Salehie, M., Mirarab, S., Tahvildari, L. Adaptive Action Selection in Autonomic Software Using Reinforcement Learning//Proceedings of International Conference on Autonomic and Autonomous Systems, Alaska, USA, 2008: 175-181.

Mao Xinjun, born in 1970, Ph.D, professor and PhD supervisor of National University of Defense Technology. His current research interests include software engineering, multi-agent system, self-adaptive and self-organizing software technology. E-mail: xjmao@nudt.edu.cn

Dong Menggao, born in 1979, PhD and assistant professor of National University of Defense Technology. His current research interests include self-adaptive system and software engineering.
E-mail:janson_mgd@163.com

Qi Zhichang, born in 1942, professor and PhD supervisor of National University of Defense Technology. His current research interests include software engineering.
E-mail: qzc@nudt.edu.cn

Yin Junwen, born in 1969, associate professor of National University of Defense Technology. His current research interests are software engineering.
E-mail: jwyin@nudt.edu.cn.

Background

More and more software systems are deployed and run in open environment like Internet. Normally, these system are open and have tightly relationships with its situated environment. The changes of environment have great impacts on the software system. For example, the events occurring in the environment may require software systems to be adjusted in either structure or behavior in order to make system more dependable, flexible, robust and reliable, etc., to satisfy the design objectives. We call such system with the capability to adjust various artifacts or attributes in response to changes in the 'self' and/or in the context that the software as self-adaptive software that are currently required in many application domains like military, enterprise, industry, etc.

Several issues should be solved when developing and running of self-adaptive systems, like software architecture, self-adaptation mechanisms and rules, design technology, programming language, supported platform, etc. In the past years, many researches have been conducted in the literatures of software engineering. These technologies provided by these researches are useful and effective to some extent to develop self-adaptive systems whose boundaries are definite (i.e., the boundaries of the systems can be clearly defined and specified at design-time) and changes are predictable and can be well-defined (i.e., the changes of environment or self can be precisely predicted and described at design-time).

However, for many complex systems like ultra-large scale system, to precisely anticipate various changes that may result in self-adaptation and predefine the complete self-adaptation requirements at design-time is extremely difficult and even impossible, because system is continuously evolving, new and maybe un-known elements of the systems may dynamically enter or leave the systems, unexpected events may occur, and the constituents of the systems are autonomous and belong to different organizations. Especially, the environment is uncontrollable and evolving. Therefore, to completely engineer the self-adaptation into the system and pre-define the set of self-adaptation events and self-adaptation logic at design-time by developer is infeasible. For example, developers can not anticipate all kinds of changes and therefore the self-adaptation logic describing how to respond to changes may evolve.

Therefore, it is a great challenge to develop self-adaptive systems in open environment, in which changes may be uncertain and unpredictable at design-time. One way to deal with the problem is to obtain changes and dynamically form self-adaptation strategy at run-time. The paper presents an approach that is based on software agent technology and organization metaphor to support the development and running of such systems. Especially, we provide approach to supporting on-line self-adaptation by introducing enforcement learning.

The research is financially supported by the National Natural Science Foundation of China under Grant Nos．61070034, 61379051；Program for New Century Excellent Talents in University with No. NCET-10-0898; and State Key Laboratory of Software Development Environment Open Fund with No. SKLSDE-2012KF-0X.