动态自适应软件体系结构重配置研究

陈向东

(复旦大学计算机科学技术学院 上海 201203) (马鞍山师范高等专科学校软件学院 马鞍山 243041)

摘 要 在当前的自适应软件研究中,人们将更多的关注点放在环境感知、服务质量建模、编程语言等方面,从而导致缺乏对自适应过程和原理的深入揭示的问题。关注体系结构,研究动态自适应过程,提出了一种软件体系结构重配置方法。该方法通过对构件、连接子的添加、删除和替换等操作来调整体系结构。基于云计算的服务器池大小动态自适应调整实验表明,动态自适应能提高系统的可信度,降低运行费用。

关键词 自适应,动态,重配置,软件体系结构,构件,连接子

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.6.040

Research on Architecture Reconfiguration of Dynamic Self-adaptive Software

CHEN Xiang-dong

(School of Computer Science, Fudan University, Shanghai 201203, China)

(College of Software, Maanshan Teacher's College, Maanshan 243041, China)

Abstract In view of the adaptive software researches in the current, people put more focus on the environment perception, quality of service modeling, programming language and so on. It results in a lack of in-depth revealing the adaptive process and principle. This paper researched on adaption from the software architecture, and put forward a kind of method of the reconfiguration of architecture in the dynamic self-adaptive process. This method adjusts the architecture by adding, deleting and updating components and connectors. The experiment of the server pool size dynamic self-adaption adjustment based on cloud computing shows that the dynamic adaption can improve the system credibility and reduce operating costs.

Keywords Self-adaption, Dynamic, Reconfiguration, Software architecture, Component, Connector

1 引言

当今时代软件面临着如下挑战:(1)应用软件赖以运行的硬件和软件环境越来越复杂。首先,随着计算机及辅助设备制造技术的发展,硬件种类越来越多且功能越来越强,从大型机到PC(个人电脑)机,再到平板电脑和智能手机,新的硬件设备层出不穷。其次,运行在应用软件上的操作系统种类及版本日益增加,结构及功能越发复杂。随着网络的普及,基于Internet的软件(即网构软件)朝着并行、异构和分布式的方向发展。这些多变的软硬件环境要求应用软件能提供足够的自适应能力。(2)用户的需求在不断变化。一方面由于用户对应用软件功能和性能的需求在提高;另一方面反映在使用软件的单位,其业务在拓展,操作软件的人员素质在提高,对应用软件不断提出新的要求。软件在运行过程中必须不断满足这些新的需求。(3)由于软硬件环境的变化,导致应用软件在持续运行中会出现一些新的缺陷或错误,这就需要软件及时地自我纠正错误以提供正确的服务,即软件自身需要具备越来越强的容错能力[1]。

为了应对由运行环境、用户需求和自身纠错等因素所导致的挑战,应用软件必须具有不断地改变和调整自己行为的能力,以适应新的环境,满足新的需求和纠正新的缺陷。早期软件都是软件设计师在设计阶段预先考虑这些变化,在系统实现前尽可能多地给出应对未知变化的设计方案。但是,系统所具有的这些有限范围的适应性都是软件的静态自适应,远远满足不了当前及今后软件在大数据、云计算和移动互联网环境下的可持续应用。目前,许多研究人员在不断探索在软件运行过程中系统如何根据环境、需求和自身的因素自我调整,以适应新的变化,这就是软件的动态自适应性[2]。动态自适应软件能够对软件自身由于环境变化所导致的错误及缺陷进行调整[3]。

软件体系结构的自适应以体系结构元素为操作对象,通过增加、删除、修改软件实体,或者调整系统的拓扑结构,来达到自适应的目的[4]。构件的自适应行为在传统的软件体系结构中是不被考虑的,但是软件实体在自适应软件体系结构中,表现出更高的抽象性、封装性和自适应性,因此在自适应体系结构中,系统结构和构件的自适应性均需要考虑[5]。

2 面向体系结构的自适应

2.1 自适应软件

自适应(Self-Adaption,SA)是指软件在运行过程中感知

到稿日期:2015-01-03 返修日期:2015-03-24 本文受2014年高等学校省级质量工程项目:软件技术专业综合改革试点(2014zy085),2015年度安徽省高等学校自然科学研究项目:动态自适应软件体系结构重配置算法及在云计算中的应用研究资助。

陈向东(1972—),男,硕士,副教授,主要研究方向为自适应软件、移动互联网。

需求变更、环境变化和自身状态，并根据感知的数据对比正常的数据，在二者不一致时对本身结构或行为进行调整，最终满足用户要求的功能和性能[6]。自适应软件(Self-Adaptive Software，SAS)即具有自适应能力的软件。自适应软件能够在整个软件生命周期中，通过检测环境的变化、需求的调整以及自身的错误或缺陷，来调整自身的结构和行为，使软件不断演变以适应外部需求和内部变化[7]。传统软件容错方法是基于冗余和多样性思想的，需要程序员事先设计多个容错版本。自适应软件系统能够在运行时通过对自身结构和行为的调整来适应运行环境变化、用户需求变更和修正自身缺陷，最终提高软件系统的可信度[8]。

2.2 软件体系结构

许多专家和学者在长期研究中从不同侧面和角度对软件体系结构(Software Architecture，SA)进行了定义，其中电气和电子工程师协会(Institute of Electrical and Electronics Engineer，IEEE)将软件体系结构抽象为由构件和连接子组成的网络[9]。

构件、连接子和配置约束是组成软件体系结构的3个主要部分。其中，构件是可重用和可预制的软件部件，它通过端口与外部进行数据交换；连接子用于构件间的连接，也是可重用和可预制的软件部件，由一组接口组成，这些接口又称角色，用于连接子与构件端口进行数据交互；配置约束是一种文件，用于描述连接子和构件之间的关联关系[10]。软件体系结构的关键模型如图1所示。

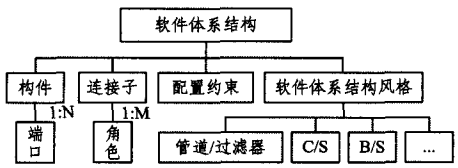

图1 软件体系结构关键模型

在软件研发初期，由于受软硬件条件的制约，加之软件在投入运行之后其环境、需求和功能基本保持不变，在软件设计和开发中，体系结构仅仅是软件开发阶段的静态描述，很少考虑到它的动态变化[11]。随着硬件的改善和面向对象等新的软件开发技术的应用，“体系结构在软件运行时需要动态变化”是大势所趋。可以通过在感知阶段获取软件体系结构信息，然后在执行阶段进行软件体系结构动态调整，从而为软件自适应提供支持。

3 自适应软件体系结构重配置机制

软件自适应性经历了从静态自适应发展到动态自适应的过程。静态自适应是软件在设计阶段，由软件设计师预先假定环境和需求的几种可能的改变而定制的程序执行流程。动态自适应是软件在运行状态下，由软件自身根据环境和需求的变化以及纠错的需要而做出的自适应调整。软件是否具有动态自适应取决于本身能否进行体系结构重配置。因此，研究体系结构重配置机制是研究软件动态自适应的关键。

3.1 动态软件体系结构

允许在系统运行过程中发生更新的体系结构被称为“动态软件体系结构”[12]。系统被创建后，“动态软件体系结构”可以在系统运行过程中，根据环境和需求的变化以及自身纠错的需要而自动进行调整。运行系统中的数据存储和运算将会因系统体系结构的动态改变而受到影响，系统因此而具有动态自适应能力。

3.2 体系结构动态更新的执行

目前支持动态体系结构机制的工具集主要是ArchStudio，它最先是由加州大学Irwine分校提出的。ArchStudio概念模型如图2所示。

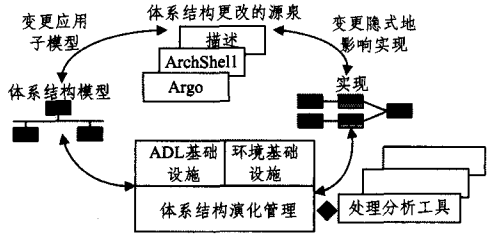

图2 ArchStudio概念模型

软件系统在运行中的改变通过一系列工具反映到它的体系结构模型上，例如图形设计环境的交互和脚本语言的改变。软件体系结构的改变包括分别对构件和连接子进行的增加、删除或修改操作，伴随这些操作，系统的拓扑结构也在发生改变。体系结构演化管理机制对系统拓扑结构的改变进行通报。对于破坏系统整体性的改变，体系结构演化管理机制有权对它进行撤销[13]。对于没有破坏系统整体性的改变，体系结构演化管理机制就相应地改变系统的执行。由此系统实现动态自适应。

3.3 基于构件的动态体系结构模型

基于构件的动态体系结构模型维护一个物化、显式、因果关联、可修改的运行时体系结构，分为3层：体系结构层、配置信息层和应用层。它是实现软件体系结构动态变化的关键[14]。该模型主要功能为：(1)为软件在运行时调整提供实时视图；(2)为判断软件运行时变化是否正确提供系统约束和属性；(3)使具体功能代码独立于软件的在线变化，实现关注点分离。基于构件的动态体系结构模型如图3所示。

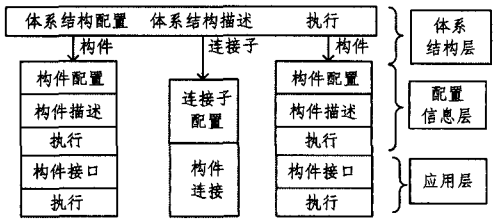

图3 基于构件的动态体系结构模型

体系结构层位于整个模型的最上层，它负责对软件整个体系结构进行控制。其中，“体系结构配置”主要实现对构件的配置、描述和执行；“体系结构描述”主要是对构件和连接子的数据以及体系结构层的行为进行描述[15]。对系统的拓扑结构、构件到处理元素之间的映射和扩展更新机制进行更改，这些都是在体系结构层完成的。

配置信息层位于模型中间，主要由构件配置、连接子配置、构件描述、执行4个部分组成。其中，“构件配置”管理构件的所有行为；“连接子配置”主要负责构件间连接和接口的配置，用于保持动态更新时的一致性；“构件描述”主要用于定义构件的结构、描述构件的功能、指明构件的版本等。在配置信息层，可以运用不同的构件装载和版本控制途径。

应用层位于整个模型最底层，由构件接口、构件连接和执行3个部分组成。其中，“构件接口”说明了构件提供的服务，如变量、消息和操作等；“构件连接”定义了构件如何与连接子相连接。在应用层，可以对构件进行添加、修改和删除操作。

以上每一层都有一个“执行”部分，它的主要功能是执行相应层的操作。这些操作包括构件和连接子的增加、删除和修改等。更新时，如有必要可以临时孤立所设计的构件。在执行更新之前，要确保两点：第一，连接子请求队列中的全部请求已被执行；第二，所涉及的构件停止发送新的请求。

3.4 算法

3.4.1 体系结构重配置算法

体系结构重配置算法如图4所示。

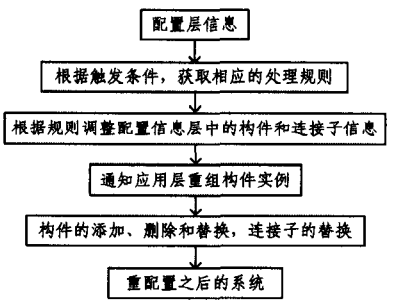

图4 体系结构重配置算法

图4中的配置信息层和应用层如3.3节所述。算法的输入为“配置层信息”，输出为“重配置之后的系统”。算法过程为：“配置信息层”从“体系结构层”抽取出构件、连接子以及它们之间的连接关系的定义，参与自适应重配置，通过触发器通知应用层并指导其重组构件和连接子实例，最终完成系统的重配置。

算法中“重组构件实例”包括“构件添加”、“构件删除”、“构件替换”和“连接子替换”，对应算法如下。

3.4.2 构件添加算法

构件添加算法如图5所示。

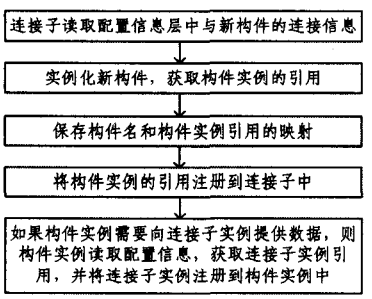

图5 构件添加算法

构件和连接子的连接信息是从配置信息层中获取得来，而不是固定写入程序代码中。构件和连接子的定义包括接口定义、连接规格定义和胶合协议。修改构件对连接子引用的通用方法由每个构件提供，该引用不能是值的集合，只能是唯一的值或者为空。同样，所有连接子也都提供了一个用于修改对构件引用的通用方法。

算法核心思想：连接子通过配置信息层获取构件实例引用的映射，实例化新构件，将新构件实例的引用注册到连接子中，完成新构件的添加。

3.4.3 构件删除算法

构件删除算法如图6所示。

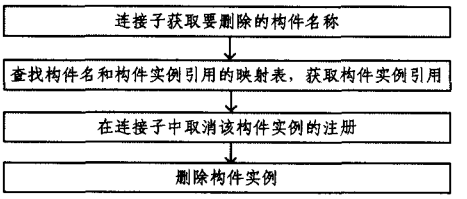

图6 构件删除算法

算法实现的关键步骤：首先获取构件实例名称和引用，再取消构件实例注册，最后删除构件实例。

3.4.4 构件替换算法

构件替换算法如图7所示。

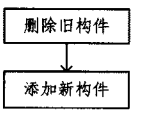

图7 构件替换算法

构件替换过程比较简单，首先删除旧构件，然后添加新构件。算法具体过程由上述的“构件添加算法”和“构件删除算法”组合而成。

3.4.5 连接子替换算法

连接子替换算法如图8所示。

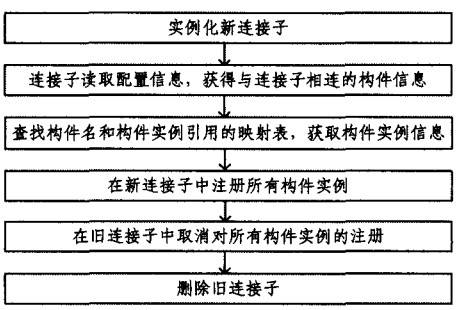

图8 连接子替换算法

算法的核心思想在于：连接子通过配置信息层查找构件实例引用映射表，首先获取相应构件实例，然后将这些构件实例注册到新连接子中，最后将旧的连接子删除，从而完成连接子的替换。

4 实验及分析

为了证明系统通过在运行时体系结构重配置来实现动态自适应的效果，设计了一个简单且有代表性的模拟实验，并在自适应和非自适应两种情况下对该系统的质量属性进行分析比较。

实验场景：基于云计算[16]的服务器池动态自适应调整策略的实现。在该场景中，动态自适应性体现在云服务[17]中：用户请求在后端通过服务器池来处理，云计算基础设施在无人工干预的情况下，根据当前负载情况，自动地调整服务器池大小，动态申请和释放计算资源，调用最少数量的服务器投入运行，最终提高系统可信度，降低运行费用。

体系结构重配置：该系统自适应体系结构如图9所示。其中，RequestProcessor(请求处理器连接子)负责外部客户端通过笔记本、平板电脑等设备的请求；ServerProxy(服务器代理构件)通过轮转方式分发客户端请求；PoolController(服务器池控制器构件)用于对指定的服务器进行激活和终止；Load Monitor(负载检测器构件)通过对日志文件进行分析来确定系统是否过载；SizeAdjustment(大小调整器)负责根据检

测结果调整服务器池大小;PoolEnlarger(策略连接子)负责在系统过载时启动新服务器的自适应机制;负载均衡器根据系统当前负载情况自动加载或卸载,使得整个系统的负载始终保持平衡状态。

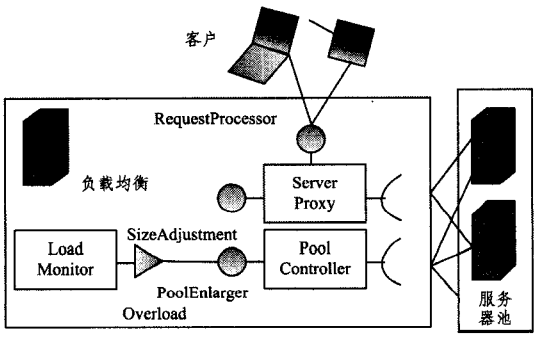

图 9 自适应服务器池体系结构

实验过程:实验中使用 3 台配置相同的计算机(主要配置参数为 Intel 四核 3400MHz CPU、DDR3 1600 4G 内存),通过 100M 以太网互相连接。服务器池上限大小为 5,服务器池初始大小为 2。每个客户端单元间隔 1s 向服务器代理构件发送请求。每个客户端请求的处理时间在 400ms～600ms 之间。某一时刻一个客户端请求只能由一个服务器单元响应。当服务器不能及时处理客户端请求时,请求将在 ServerProxy 构件上排队等候。Load Monitor 构件采样频率为 10s 一次,当请求响应时间超过 1s 时,输出 Overload 上下文事件,并且激活策略连接子 PoolEnlarger。整个实验在 10min 内完成,初始客户数为 8,在第 200s 和第 400s 时分别向系统动态添加 4 个客户端,从而模拟动态增加负载的情景。

实验结果及分析:测试结果如图 10 所示。由实验结果可知,当响应时间超过阈值时,服务器池将会自动扩大,这样可以保证及时处理客户端请求。实验过程中,为了验证软件自适应对质量属性的改变,证明自适应带来的收益,同时对非自适应情况也做了测试,结果如图 11 所示。将自适应和非自适应两种情况进行对比,结果表明,非自适应服务器池在资源增加时性能下降了,在高负载情况下,请求响应的时间无法控制在 1000ms 以内。实验结果证明,通过体系结构重配置实现的动态自适应所带来的效益是非常明显的。

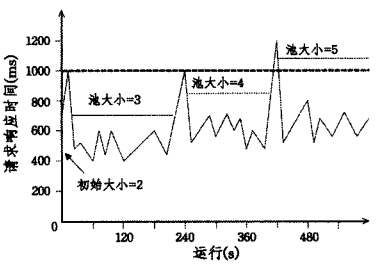

图 10 服务器池自适应测试结果

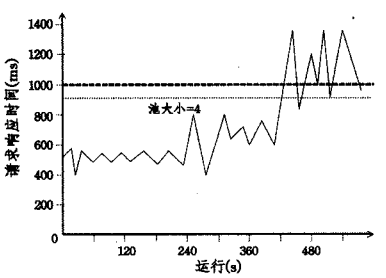

图 11 服务器池非自适应测试结果

结束语 随着软件技术的进步和研究的深入,自适应研究领域越来越广。总体来说,自适应主要研究方向包括环境感知、服务质量(Quality of Serve,QoS)建模、自适应参数调整、自适应可信度保证、基于 Agent 的自适应、面向普适计算的自适应、基于本体的自适应等方面[18]。之前许多自适应研究人员将更多关注点集中在系统之外的环境信息方面,对这些数据进行收集、处理和分析,并根据用户需求对服务质量建立数学模型等。从调整自适应目标和明确自适应触发机制方面来看,这些研究有一定的帮助作用。与此同时,在自适应研究中仍然缺乏对自适应过程和原理的探究。自适应研究最核心的问题是在自适应过程中,软件体系结构如何在系统运行过程中实现重配置。近年来,虽然也有从体系结构入手研究自适应的,但是很多研究存在一些局限性。典型的有:将体系结构与运行系统分离,在系统运行过程中,另外借助特殊的措施收集体系结构信息,如增加体系结构收集器等。虽然这些措施在一定程度上也能实现系统自适应性,但无形中增加了系统自适应实施的成本和机会,使得体系结构始终不能作为运行系统的具体组成部分,而只能是抽象。当信息从分离的体系结构中被提取时,它们无法反映真实的需求变更,最终有可能导致错误的适应行为。

在动态、多变的环境下,为满足环境开放、成员异构、行为主动的基于 Internet 的软件(网构软件)提供高质量、稳定的服务的需求,本文在运行系统中融合体系结构,面向动态自适应软件,提出一种体系结构重配置的方法,指导针对可信度和性能的软件自适应。通过对构件和连接子的添加、修改和删除等操作来实现体系结构的调整。由于软件中构件和连接子都是具体的实体,因此该方法可操作性强。实验中通过运行时刻实施软件体系结构的修改,最终保证系统的可信度和性能。

未来需要进一步研究的方面还有很多。首先是如何将可信度和性能指标进一步细分,然后分配给体系结构中的相应构件和连接子等构成元素,这样就可以通过对构件和连接子的操作来调整相应的可信度和性能指标,达到灵活地进行自适应重配置的目的。其次,对于大型软件系统,其体系结构建模过程是多阶段的、复杂的,如何将自适应策略从高层体系结构模型转化到低层体系结构模型中,要进一步探索。另外,针对本文的理论研究,构建更多的应用场景,做更多的实验以获取大量的测试数据进行反复论证,这也是今后需要深入开展的工作。

参 考 文 献

[1] Ruan Fei, Wang Yuan. Development Analysis of Information Grid Technologies[J]. Journal of China Academy of Electronics and Information Technology, 2011(2): 165-169

[2] Yu Chun, Ma Qian, Ma Xiao-xing, et al. An Architecture-oriented Mechanism for Self-adaptation of Software Systems[J]. Journal of Nanjing University (Natural Sciences), 2006, 42(2): 120-130

[3] Ding Bo. Research on Key Techniques of Software Self-Adaptation[D]. Hunan: National University of Defense Technology, 2010

[4] Mei Hong, Shen Jun-rong. Progress of Research on Software Architecture[J]. Journal of Software, 2006, 17(6): 1257-1275

(下转第 215 页)

参 考 文 献

[1] Pawlak Z. Rough set[J]. Communications of the ACM, 1995, 38 (11): 89-95

[2] 王国胤. ROUGH 集理论与知识获取[M]. 西安: 西安交通大学出版社, 2001
Wang Guo-yin. Rough Set Theory and Knowledge Acquisition [M]. Xi'an: Xi'an Jiaotong University Press, 2001

[3] Pawlak Z, Skowron A. Rudiments of rough sets[J]. Information Sciences, 2007, 177: 3-27

[4] 钱文斌, 徐章艳, 黄丽宇, 等. 基于信息熵的二进制差别矩阵属性约简算法[J]. 计算机工程与应用, 2010, 46(6): 120-123
Qian Wen-bin, Xu Zhang-yan, Huang Li-yu, et al. Attribution reduction algorithm based on binary discernibility matrix of information entropy[J]. Computer Engineering and Applications, 2010, 46(6): 120-123

[5] Skowron A, Rauszer C. The discernibility matrices and functions in information systems[C]// Slowiński R, ed. Intelligent Decision Support, Handbook of Applications and Advances of the Rough Sets Theory. Kluwer, Dordrecht, 1992, 11: 331-362

[6] Hu X H, Cereone N. Learning in Relational Database: A Rough Set Approach[J]. International Journal of Computational Intelligence, 1995, 11(2): 323-338

[7] Yamaguchi D. Attribute dependency functions considering data efficiency[J]. International Journal of Approximate Reasoning, 2009, 51(1): 89-98

[8] Felix R, Ushio T. Rough sets-based machine learning using a binary discernibility matrix[C]// Proceeding of 2nd International Conference on Intelligent Processing and Manufacturing of Materials. Ha-waii, 1999: 299-305

[9] 支天云, 苗夺谦. 二进制可辨矩阵的变换及高效属性约简算法的构造[J]. 计算机科学, 2002, 29(2): 140-142
Zhi Tian-yun, Miao Duo-qian. The binary discernibility matrix's transformation and high efficiency attributes reduction algorithm's conformation[J]. Computer Science, 2002, 29(2): 140-142

[10] 李龙澍, 王慧萍, 徐怡. 二进制可分辨矩阵的最小属性约简算法[J]. 计算机技术与发展, 2010, 20(6): 93-96
Li Long-shu, Wang Hui-ping, Xu Yi. Algorithm for the least attribute reduction of binary discernibility matrix[J]. Computer Technology and Development, 2010, 20(6): 93-96

[11] 陈宸, 赵军. 一种新的基于二进制分辨矩阵的属性约简方法[J]. 计算机应用与软件, 2013, 30(9): 123-127
Chen Chen, Zhao Jun. A new attribute reduction method based on binary discernibility matrix [J]. Computer Applications and Software, 2013, 30(9): 123-127

[12] 徐章艳, 杨炳儒, 宋威. 基于简化的二进制差别矩阵的快速属性约简算法[J]. 计算机科学, 2006, 33(4): 155-158
Xu Zhang-yan, Yang Bing-ru, Song Wei. Quick attribution reduction algorithm based on simple binary discernibility matrix [J]. Computer Science, 2006, 33(4): 155-158

[13] 任倩, 罗月童. 一种二进制可分辨矩阵修正方法及其求核[J]. 小型微型计算机系统, 2013, 34(6): 1437-1440
Ren Qian, Luo Yue-tong. An new method for modifying binary discernibility matrix and computation of core[J]. Journal of Chinese Computer Systems, 2013, 34(6): 1437-1440

[14] 蒙祖强, 史忠植. 一种新的基于简化二进制可辨矩阵的相对约简算法[J]. 控制与决策, 2008, 23(9): 976-978
Meng Zu-qiang, Shi Zhong-zhi. Algorithm for relative reduction based on simplifried binary discernibility matrix[J]. Control and Decision, 2008, 23(9): 976-978

[15] 桂现才. 简化的二进制差别矩阵属性约简算法的改进[J]. 计算机工程与设计, 2007, 28(16): 3971-3973
Gui Xian-cai. Improved algorithm for attribute reduction based on simple binary discernibility matrix[J]. Computer Engineering and Design, 2007, 28(16): 3971-3973

[16] 杨传健, 葛浩, 李龙澍. 垂直划分二进制可分辨矩阵的属性约简[J]. 控制与决策, 2013, 28(4): 563-568
Yang Chuan-jian, Ge Hao, Li Long-shu. Attribute reduction of vertically partitioned binary discernibility matrix [J]. Control and Decision, 2013, 28(4): 563-568

[17] Wang Jue, Wang Ju. Reduction algorithms based on discernibility matrix: the ordered attributes method[J]. Journal of Computer Science and Technology, 2001, 16(6): 489-504

(上接第 188 页)

[5] Li Bing. Research on Key Technology of Self Adaptive Software [D]. Jilin: Jilin University, 2012

[6] Hu Qing-lei, Xiao Bing. Adaptive fault tolerant control using integral sliding mode strategy with application to flexible spacecraft [J]. International Journal of Systems Science, 2013, 44 (12): 2273-2286

[7] Zhang You-sheng. Software architecture [M]. Beijing: Tsinghua University press, 2006

[8] Li Jin-gang, Zhao Shi-lei, Du Ning. The theory and application of software architecture [M]. Beijing: Tsinghua University press, 2013

[9] Zhou Su, Peng Bin, Zhang Yong, et al. Software architecture and design [M]. Beijing: Tsinghua University press, 2013

[10] Chen Xiang-dong. New system based on event-driven and service-oriented business activity monitoring design and implementation[J]. Application Research of Computers, 2012, 29(3): 977-980

[11] Tang Shan, Li Li-ping, Tan Wen-an. Research on Runtime Monitoring for Self-adaptive and Reconfigurable Software Systems [J]. Computer Science, 2013, 40(11): 191-196

[12] Gao Jun, Shen Cai-liang, Zheng Mei-fang et al. Architecture description language of software oriented self-adaptive[J]. Application Research of Computers, 2010, 27(5): 1796-1801

[13] Chen Xiang-dong. Characteristics and Application of New Generation Websites System[J]. Journal of Beihua University(Natural Science), 2011, 12(3): 359-362

[14] Li Li, Liao Jian-wei, Ou Ling. Clound computing: an introduction [J]. Application Research of Computers, 2010, 27(12): 4419-4422

[15] Chen Quan, Deng Qian-ni. Clound computing and its key techniques[J]. Journal of Computer Applications, 2009, 29(9): 2562-2567

[16] Pan Jian, Zhou Yu, Luo Bin, et al. An Ontology-based Software Self-adaptation Mechanism [J]. Computer Science, 2007, 34 (11): 264-269

[17] Ding Bo, Wang Huai-min, Shi Dian-xi. Pervasive middleware technology [J]. Journal of Computer Science and Frontiers, 2007, 1(3): 241-254

[18] Liu Hui, Shi Dian-xi, Liu Ming, et al. Oriented Self-Adaptive Software Integrated Environment[J]. Computer Engineering & Science, 2010, 32(1): 105-108