一种基于本体的软件自适应机制*)

潘 健 周 宇 罗 滨 马晓星 吕 建

(南京大学计算机软件新技术国家重点实验室 南京大学计算机软件研究所 南京 210093)

摘 要 为维持一定的可用性和服务质量,许多软件系统需要动态地自我调整,以适应环境和需求的变化。本文提出了一种基于本体的软件自适应机制,通过本体建模来认识和表达分布在问题空间和解空间中的与软件自适应相关的要素,并采用定义良好的推理规则来操纵决策要素,从而实现了问题空间和解空间在软件自适应中的有机结合。在此基础上,本文设计了基于本体的自适应软件结构并在原型系统中加以实现,最后通过一个应用实例验证了基于本体的软件自适应机制的有效性。

关键词 自适应,软件体系结构,本体,需求

An Ontology-based Software Self-adaptation Mechanism

PAN Jian ZHOU Yu LUO Bin MA Xiao-Xing LU Jian

(National Key Laboratory for Novel Software Technology, Institute of Computer Software, Nanjing University, Nanjing 210093)

Abstract Existing approaches for self-adaptive software systems often have inadequate support for the understanding and reasoning about the links between the problem space and the solution space. In order to bridge this gap, an ontology-based self-adaptation mechanism is proposed. It divides factors which are critical to system evolution into three categories: requirement-related, architecture-related and application-related. These factors are reified by building corresponding ontology. A set of ontology-based rules are defined to reason about these factors and give appropriate system evolution prescriptions. The mechanism is implemented in a prototype system and a sample application is developed to illustrate its effectiveness.

Keywords Self-adaptation, Software architecture, Ontology, Requirements

1 引言

随着软件系统的规模、复杂度的增加及其运行环境的转变,仅仅依靠手工调整已经无法保证软件系统的质量。人们开始研究在运行过程中能够不断评价自身行为质量,并在其行为无法满足用户需求时主动调整自身行为的软件系统,即所谓的自适应软件[1,2]。由于软件体系结构在软件工程实践中所起的作用非常重要,且在软件体系结构层面上实施自适应具备很多优点,因此基于体系结构的软件自适应成为软件自适应研究的一个热点课题[3,4]。

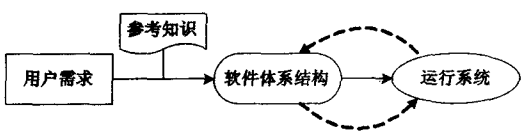

图1 基于体系结构的自适应软件开发

现有的基于体系结构的自适应软件开发如图1所示,它表现为一个从问题空间到解空间的单向转换过程(图中实线)。自适应软件通过运行时刻的监控来评价自身行为质量,当软件体系结构的某些约束被破坏时,触发自适应动作调整软件体系结构和运行系统(图中虚线环)。现有研究的不足在于,对问题空间及其转化过程在软件自适应中的作用强调不够。问题空间中的用户需求是判断软件系统是否需要自适应的根本依据,也是检验自适应效果的唯一标准;用户需求到软件体系结构转化过程中用到的参考知识对于最终得到的软件体系结构有很大的影响。两者的缺失,使得它们的变化无法直接作用于软件自适应。带来的可能后果是,软件的自适应调整会使软件系统沿着背离用户需求的方向演化。已经有学者注意到了这方面的问题并做了一定的尝试[5],但较为系统的工作还未见。

把问题空间引入软件自适应,需要解决以下3个问题:决策要素的认识,决策要素的表达和推理规则的组织。软件系统的问题空间和解空间包含的知识非常丰富,而自适应所依赖的只是其中的一部分,因此需要从中抽取和软件自适应相关的知识,即所谓的决策要素。由于决策要素的载体来自软件工程的不同阶段,因此这些载体对决策要素的表达方式一般不同。显然,在自适应软件中直接利用这些决策要素是非常困难的,因此需要研究一个描述框架来为决策要素提供统一的表达方式,从而为决策要素的操纵创造条件。软件系统的自适应决策是一个将自适应规则作用于决策要素的推理过程,因此如何组织、描述推理规则是非常关键的。而表达准确,容易理解,可维护性强且具备一定的可扩展性是推理规则

*)本文工作受国家973(2002CB312002)、国家自然科学基金项目(60403014,60233010)和江苏省自然科学基金(BK2006712)资助。潘 健 硕士研究生,主要研究兴趣为Internet软件技术;周 宇 博士研究生,主要研究兴趣为Internet软件技术;罗 滨 硕士研究生,主要研究兴趣Internet软件技术;马晓星 副教授,主要研究兴趣为Internet软件技术、软件体系结构;吕 建 博士,教授,博士生导师,主要研究领域为软件自动化、并行程序形式化方法、面向对象语言和环境。

的基本要求。

针对上述问题，我们对自适应软件的决策要素作了研究，我们认为应该透过决策要素的各种原始表达形式，从决策要素的概念本质和概念之间的关联出发去认识和表达决策要素。计算机研究中的本体是在对某一给定领域的知识实现共同理解并予以概念化后给出的显式的形式化的规约说明[6]。本体非常适用于表达决策要素的概念本质和概念之间的关联，基于本体的推理规则亦可以满足自适应规则的需求，因此我们在已有工作[4,7]的基础上提出了一种基于本体的软件自适应机制。我们采用本体对决策要素建模，以 OWL(Web Ontology Language)[8] 作为本体描述语言，结合 RDF(Resource Description Framework)[9] 来描述自适应软件的决策要素，从而确保这些要素在一个统一的描述框架下被操纵。我们采用定义在 OWL 和 RDF 上的推理规则来描述自适应逻辑，这些规则和一阶谓词逻辑规则相似，具备较强的表达能力。同时，我们基于 Jena[10] 实现了自适应决策引擎，通过决策引擎把推理规则作用于决策要素，从而给出合理的自适应动作调整软件系统。

接下来，我们将在第 2 节介绍基于本体的自适应软件结构，从而揭示基于本体的软件自适应机制的基本原理；第 3 节讨论基于本体的自适应决策，主要包括决策要素的认识、表达以及推理规则的组织；第 4 节讨论基于 Artemis-MAC 系统实现的一个简单的自适应软件应用实例，以验证基于本体的软件自适应机制的有效性；第 5 节介绍相关工作，并和我们的工作做一定的比较；最后小结全文。

2 基于本体的自适应软件结构

基于本体的自适应软件总体结构如图 2 所示。

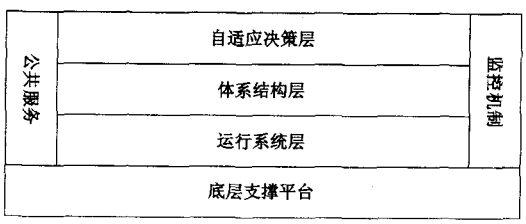

图 2 基于本体的自适应软件总体结构图

整个系统由 5 个部分组成，分别是：运行系统层、体系结构层、自适应决策层、监控机制和公共服务。底层支撑平台是通用的软件运行基础设施，包括硬件、操作系统、网络设施等。从完整性角度考虑，我们把底层支撑平台放在系统的总体结构里面，但它不在我们的讨论范围之内。基于本体的自适应软件和其他基于体系结构的自适应软件的不同点在于，决策要素的存储和使用从它的生成者和收集者中分离出来，由系统中新增的自适应决策层来统一管理和操纵。当然，自适应决策层的增加也会使系统其他部分的功能发生变化。下面简要介绍一下各个组成部分。

(1) 运行系统层

运行系统层是执行用户计算任务的模块。在 Internet 计算环境下，它往往是由网络中分布在多个节点上的软件服务按照体系结构层的配置约束通过某些标准的协议集成在一起构成的。为了支持软件自适应，运行系统层还需要提供信息采集接口和系统调整接口。信息采集接口提供给监控机制使用，以便反映运行系统层的实时状态；系统调整接口由自适应执行引擎调用，负责根据自适应决策调整运行系统。

(2) 体系结构层

体系结构层从系统全局的角度维护当前系统配置状态，并对系统级属性的监控提供支持。因此体系结构层不仅包括当前系统的静态描述信息，还包括运行时刻体系结构相关的信息。体系结构层由两个部分组成：体系结构模型和体系结构管理器。体系结构模型维护系统实时的体系结构信息，包括系统的拓扑结构、构件的属性等。以往的基于体系结构的软件自适应研究发现[11]，对系统的动态调整可以在体系结构的抽象层面上表达。体系结构管理器就是负责根据自适应决策把对系统的动态调整表达在体系结构层面上，从而间接达到调整运行系统的目的。体系结构管理器实现了基于本体的自适应软件的自适应执行引擎的功能，此外它还为监控机制提供信息采集支持。实现时，我们把体系结构层的两个模块映射成一个运行时体系结构对象，其中体系结构模型作为对象的属性，体系结构管理器作为对象的行为。

(3) 自适应决策层

自适应决策层负责维护和管理自适应软件的决策要素和自适应逻辑，并依据系统的运行状况给出合理的系统调整策略。该层的结构如图 3 所示。

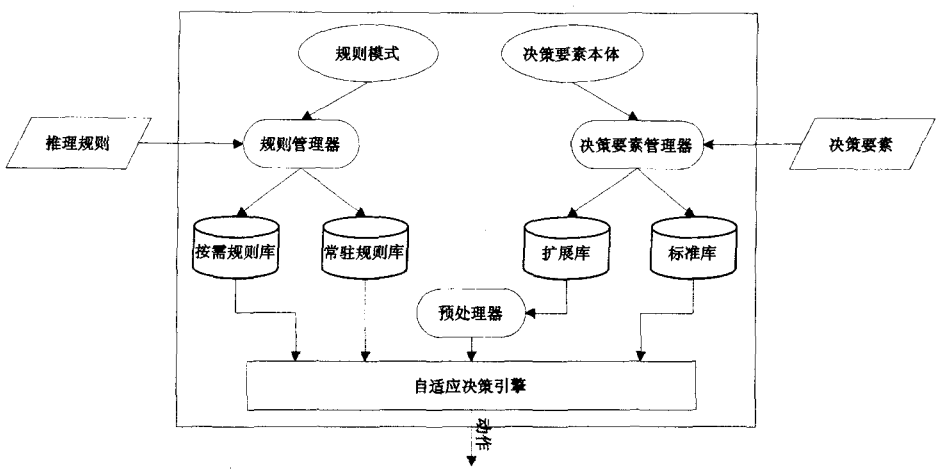

图 3 自适应决策层结构图

该层以决策要素和推理规则为输入，系统调整动作为输出。从图3可以看出，自适应决策层由3个主要模块构成：决策要素管理模块，自适应逻辑管理模块和自适应决策模块。

决策要素管理模块维护整个系统的决策要素。决策要素管理器接收来自监控机制的决策要素数据，用决策要素本体规定的术语集描述决策要素，并根据该决策要素的特点把它存入相应的数据库。这里我们设计了两类决策要素数据库：标准库和扩展库。标准库存放采用标准RDF三元组表示的决策要素，扩展库存放采用扩展的RDF元组表示的决策要素。扩展库的存在主要是为了处理一些RDF三元组无法或者难以表示的决策要素，增加系统的灵活性。

自适应逻辑管理模块维护自适应决策所需的推理规则。规则管理器接收来自用户的规则数据，同时采用规则模式对它进行校验。规则模式用于保证输入的规则符合一定的语法，这个将在下文详述。我们把推理规则分为两类：常驻规则和按需规则。常驻规则参与每一次自适应决策；按需规则只在与其相关的要素发生变化时参与自适应决策。这样可以有效减小自适应决策过程中用到的规则集的规模，从而提升自适应决策的性能[12]。

自适应决策模块包括预处理器和自适应决策引擎。为了保证自适应决策引擎的通用性和高效性，它仅支持对标准RDF三元组的推理。为了能够在自适应决策时使用扩展库中的数据，我们设计了预处理器。预处理器根据用户指定的处理算法把扩展库中的数据处理成标准的RDF三元组，然后和标准库中的数据一起交给自适应决策引擎做推理。

(4) 监控机制

监控机制负责采集系统信息。和以往的监控机制不同的是，我们的监控机制只采集信息，不消费信息。因此，我们的监控机制只包含Probe和Gauge两个部分。Probe采集原始数据，Gauge对Probe传来的数据做一定的处理。关于它们的详细论述，可以参考文[7]。

(5) 公共服务

公共服务是用于辅助自适应软件构造和运行的软件设施。它们一般独立于具体的应用场景。基于本体的自适应软件的公共服务中最重要的一个是注册/查询服务。我们的决策要素是采用OWL/RDF描述的，决策要素中不能直接保存运行系统相关构件的引用。因此需要在决策要素实体和构件实体之间建立一种映射关系。虽然我们可以通过程序代码建立这种映射关系，但是这样带来的后果是结构固化，缺乏灵活性。因此，我们采用注册/查询的方式。在系统组装的时候，把决策要素实体和构件实体注册到中心数据库，并定义它们之间的映射关系。在使用的时候，系统可以利用注册/查询服务提供的接口获取决策要素实体和构件实体的对应关系。同时，利用注册/查询服务提供的接口可以很方便的增加、删除、更新决策要素实体、构件实体及相关的映射关系。

综上所述，基于本体的自适应软件通过增加自适应决策层，进一步分离了自适应部件和业务逻辑部件，在一定程度上简化了自适应软件的开发，提高了自适应软件的可复用性和可维护性。

3 基于本体的自适应决策

基于本体的软件自适应机制的核心是基于本体的自适应决策。这涉及3个方面的工作：决策要素的认识，决策要素的表达和推理规则的组织。下面我们将逐一介绍。

3.1 自适应软件的决策要素

决策要素是自适应决策的依据，决策要素选取得恰当与否将直接影响到软件自适应决策的合理性。从来源看，自适应软件的决策要素可以分为需求相关要素、体系结构相关要素和应用相关要素。需求相关要素主要是用户需求中和软件自适应相关的部分，用于指导和校验软件自适应。软件系统的体系结构一般是指该系统的构件组成、构件之间的相互关系和连接方式，所形成的系统结构配置，及其动机、模式和约束。因此体系结构相关要素描述当前系统的组织结构和系统中相关构件的状态，这些是评估软件系统行为的重要依据。体系结构相关要素相对抽象一些，不包括一些具体的部署信息、平台相关的信息，这些由应用相关要素来描述。这里的应用相关要素除包含运行平台的一些信息外，也包含了一些与特定应用相关的信息，比如需求分析过程中用到的参考知识。

3.2 基于本体的决策要素建模

从3.1可知，自适应软件的决策要素的来源和内容呈多样化的特点。如何规范、准确的描述决策要素，为自适应决策提供良好的支持是我们研究的重点。我们认为一个显式的决策要素模型有助于解决上述问题。事实上，一个好的决策要素模型不仅可以帮助我们规范的描述决策要素，也有利于不同系统的决策要素的共享和集成，从而为多个自适应软件系统的集成和互操作打下基础。决策要素是多种信息的集合，我们建模的目的在于使这些信息实现共同理解，并予以概念化，然后用规范的显式的规约来说明这些概念。因此，我们采用基于本体的建模方式，通过构造决策要素本体来描述决策要素模型。下面我们将分三个部分讨论3.1中提出的三类决策要素的建模。

3.2.1 需求相关要素建模

我们对需求相关要素建模的目的在于从需求分析结果中抽取和软件自适应相关的知识，因此我们只对需求分析的部分知识建模。我们从面向目标的需求工程(Goal-Oriented Requirements Engineering)[13]的研究成果中发现用户需求中的目标是需求相关要素建模的对象。对目标的建模主要考虑以下3个方面内容：

- 目标类型及类别。目标一般分为两大类：功能性目标和非功能性目标。它们还可以进一步细分。
- 目标属性。目标的属性主要有名称、描述、优先级、有用性和可行性等。
- 目标链接。目标链接用于刻画目标之间、目标与其他非目标元素之间的关系。需求相关要素建模主要关注两种定义在目标之间的链接：AND链接和OR链接。

综上，我们可以得到如图4所示的需求相关要素的部分本体层次关系图。

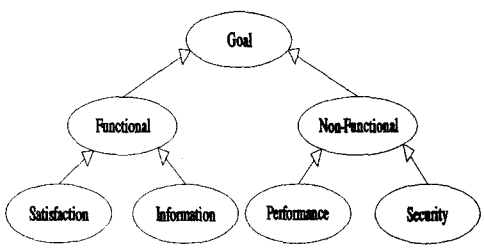

图4 需求相关要素本体

3.2.2 体系结构相关要素建模

软件体系结构从相对抽象的层面描述当前系统的构件组

成、构件之间的相互关系和连接方式，它是软件系统相关各方理解系统的钥匙，是沟通问题空间和解空间的最佳候选。目前研究中我们以 Acme[14] 作为基本的体系结构描述语言。我们认为 Acme 用于描述体系结构的最基本的元素是构件、连接子、端口和角色。因此我们对体系结构相关要素建模主要考虑这 4 个元素：

• 构件描述系统的基本计算单元和数据存储单元。典型的构件包括客户端、服务器、过滤器等。构件的状态由它的属性描述。

• 连接子描述构件间相互作用的连接机制，通常是构件间的通信和协调活动。典型的连接子包括管道、过程调用、事件广播等。连接子的状态由它的属性描述。

• 端口描述构件与其环境交互的接口。

• 角色描述连接子的界面元素或接口。

对特定应用领域来说，其系统组织方式往往存在着某些共性，通过总结这些共性，可以得到某些惯用的模式，称为体系结构风格。体系结构风格为设计人员的交流提供了公共的术语空间，促进了设计复用和代码复用。因此，体系结构相关要素模型同样应该支持体系结构风格，以提高体系结构相关要素模型的复用性。体系结构风格的相关内容可以参考文[15]，这里不再赘述。综上，我们可以得到如图 5 的体系结构相关要素的部分本体层次关系图。

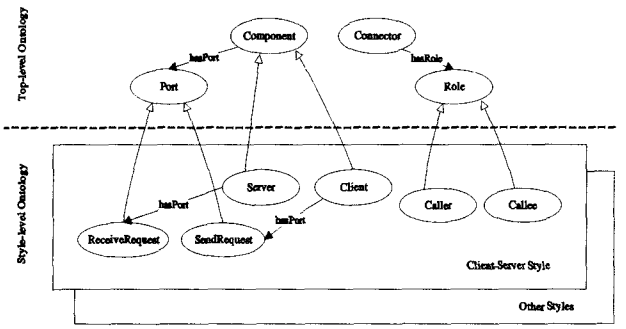

图 5　体系结构相关要素本体

3.2.3　应用相关要素建模

应用场景中包含的信息非常多，基于本体的自适应软件的应用相关要素一般只关注和软件自适应相关的信息。应用相关要素具有一定的分散性，它可能是需求分析依赖的某些参考知识，也可能是部署阶段的某个硬件参数。因此，应用相关要素建模贯穿需求分析、软件设计和部署这 3 个阶段。由于建模得到的应用相关要素本体和应用场景密切相关，所以我们无法给出一个应用相关要素本体的大框架，应用相关要素本体需要根据具体的应用场景来设计。

3.3　基于本体的自适应逻辑

在我们基于本体的自适应机制中，决策要素本体采用 OWL 描述，决策要素采用 RDF 描述。我们的自适应决策引擎是基于 Jena 的推理引擎实现的，因此我们的自适应逻辑是由定义在 OWL/RDF 上的推理规则组成的。推理规则的主要语法描述如下。

```
〈rule〉      ::=〈bare-rule〉"."|"["〈bare-rule〉"["|"["〈rulename〉
             ":"〈bare-rule〉"]"
〈bare-rule〉 ::=〈term〉{","〈term〉}"->"〈hterm〉{","〈hterm〉}
             |〈term〉{","〈term〉}"<-"〈term〉{","
             〈term〉}
〈hterm〉     ::=〈term〉|"["〈bare-rule〉"]"
〈term〉       ::="("〈node〉","〈node〉","〈node〉")"|"("〈node〉","
             〈node〉","〈runctor〉")"
             |〈builtin〉"("〈node〉{","〈node〉}")"
〈functor〉    ::=〈functorname〉"("〈node〉{","〈node〉}")"
〈node〉       ::=〈uri-ref〉|〈prefix〉":"〈localname〉|"?"〈varname〉|
             ""〈literal〉""
             |"'"〈lex〉"'~"〈typeURI〉|〈number〉
```

上述语法定义采用类 BNF 表示法。这个语法定义并不完备，其中没有定义的〈rulename〉、〈functorname〉、〈prefix〉、〈localname〉、〈varname〉、〈literal〉、〈lex>〉字符串，〈builtin〉是 Jena 的内置函数的名字，〈uri-ref〉和〈typeURI〉是 URI(Universal Resource Identifier)字符串，〈number〉是整型或浮点型数字。

综上所述，决策要素本体可以统一表达自适应软件的决策要素，为问题空间和解空间在软件自适应中的结合提供了基础；自适应推理规则实现了问题空间和解空间对软件自适应的指导和校验。决策要素本体和自适应推理规则的有机结合保证了自适应决策的合理性。

4　应用实例

Artemis-MAC 是南京大学计算机软件研究所在研究基于 Agent 的软件协同新技术过程中开发的一个原型系统，对它的进一步介绍可以参考文[16]，这里不再赘述。我们在其中实现了基于本体的软件自适应机制。为了验证该机制的有效性，我们基于 Artemis-MAC 开发了一个简单的网络订票系统。

4.1　实例简介

随着电子商务日益普及，越来越多的消费者开始网上购买物品和服务。我们的网络订票应用就是在这样的背景下开发的。我们设计的是一个综合的网络票务服务平台，提供诸如汽车票、火车票等多种车票的发售。这些车票的订购服务由各家票务公司提供，平台提供接人接口。为了使我们的票务服务更加贴心，系统将会记录用户的访问历史和一些个性化的信息。用户登录系统操作时，系统会根据他的访问历史和个人喜好推荐合适的票务服务。显然，响应速度快是该系统的一个基本需求。

在对上述应用场景分析后我们发现 Master/Slave 体系结构风格是用来构建系统的最佳选择。其中 Master 分发用户请求，Slave 提供某种具体的票务服务。为了能够给用户推荐合适的票务服务，Master 在分发用户请求之前，需要根据用户的访问历史和个人喜好做相关的计算。显然，该系统的响应时间由两个方面构成：Master 的响应时间和 Slave 的响应时间。为了研究的需要，我们对场景做了简化：假设 Slave 的响应速度始终足够快，而 Master 的响应速度是系统的瓶颈。这个假设的合理性在于：Master 需要为每个用户推荐票务服务并分发请求，而 Slave 只提供一个公司的一种票务服务，它只为部分用户服务。

系统按照 Master/Slave 体系结构风格搭建，搜索组装相应的构件服务之后，生成具有该体系结构风格的体系结构对象，并发送到各参与节点。同时初始化自适应部件，这个过程包括载入决策要素本体，设置初始的决策要素信息，配置自适应逻辑及启动监控机制等。下面我们将给出和该实例相关的决策要素及推理规则的部分代码，然后展示其运行效果。

4.2　相关代码

(1)需求相关要素

在我们这个场景中，一个很自然的需求就是网站的响应速度要足够快。描述这个目标的代码如下所示：

```
〈Non-Functional rdf:ID="goal_3"〉
    〈name rdf:datatype=
```

```
    "http:www. we. org/2001/XML Schema#string">
    quickResponse</name>
    <specification rdf:datatype=
    "http://www. w3. org/2001/XMLSchema#string">
        Users can get their response quickly.
    </specification>
    <priority rdf:datatype=
    "http://www. w3. org/2001/XMLSchema#int">
    5</priority>
    <utility rdf:datatype=
    "http://www. w3. org/2001/XMLSchema#double">
    0. 92 </utility>
        <feasibility rdf:datatype=
        "http://www. w3. org/2001/XMLSchema#double">
        0. 85</feasibility>
        </Non-Functional>
```

(2)应用相关要素

上面我们只是给出了一个目标,但是没有对"响应快"进行量化。对于快的定义是需求分析中用到的一个参考知识,需要根据当前网民的上网体验来确定。这里我们假设目前网民普遍认为一个速度快的网站的响应时间应该在2000毫秒以内。描述这个应用相关的决策要素的代码如下所示:

```
<Reference rdf:ID="reference_1">
    <name rdf:datatype=
    "http://www. w3. org/2001/XMLSchema#string">
    quickPesponse</name>
    <description rdf:dataatype=
    "http://www. w3. org/2001/XMLSchema#string">
    If a web site is quick,its response time should less than this value.
    </description>
    <value rdf:datatype=
    "http://www. w3. org/2001/XMLSchema#int">
    2000</value>
</Reference>
```

(3)体系结构相关要素

为了满足用户需求,我们需要考虑 master 和 slave 的响应速度。由于我们假设 slave 的响应速度足够快,因此我们只需要考虑 master 的影响。描述 master 的代码如下所示:

```
<Master rdf:ID="master_1">
   <name rdf:datatype=
   "http://www. w3. org/2001/XMLSchema#string">
   master1</name>
   <description rdf:datatype=
   "http://www. w3. org/2001/XMLSchema#string">
       This is a master. It does some computation and delivers tasks
       to the appropriate slaes.
   </description>
   <response Time rdf:datatype=
   "http://www. w3. org/2001/XMLSchema#double">
   1230</responseTime>
   </Master>
```

(4)推理规则

有了上述决策要素后,我们需要借助推理规则把它们联系起来,从而为软件的自适应调整服务。该实例使用的推理规则如下所示:

```
@prefix ns:<http://ics. nju. edu. cn/artemis-mac#>
@prefix xsd:<http://www. w3. org/2001/XMLSchema#>
[r0:(? goal,rdf:name,'quickResponse'),
    (? goal,ns:name,'quickResponse'),
    (? reference,rdf:typ,ns:Reference),
    (? reference,ns:name,'quickResponse'),
    (? reference,ns:value,? value),
    (? master,rdf:type,ns:Master)->
    (? master,ns:maxResponseTime,? value)]
[r1:(? master,rdf:type,ns:Master),
    (? master,ns:responseTime,? time),
    (? master,ns:maxResponseTime,? max),
   greaterThan(? time,? max),
    (? master,ns:hasAction,? action)->
    (? action,ns:name,'addMater'),
    (? action,ns:actOn,? master)]
```

该应用的自适应逻辑由上述两条推理规则构成,其中 r0 规则根据需求相关要素和应用相关要素把用户需求映射成体系结构约束;r1 规则根据体系结构相关要素的当前状态及其约束推导出合理的自适应调整动作。

4.3 运行效果

这个应用的服务节点分布在一个由3台通过100Mb/s的快速以太网连接的双 Xeon 2.8G CPU 工作站构成的局域网上。我们通过模拟客户端不断发送 http 请求来触发自适应。系统变化如图6所示。

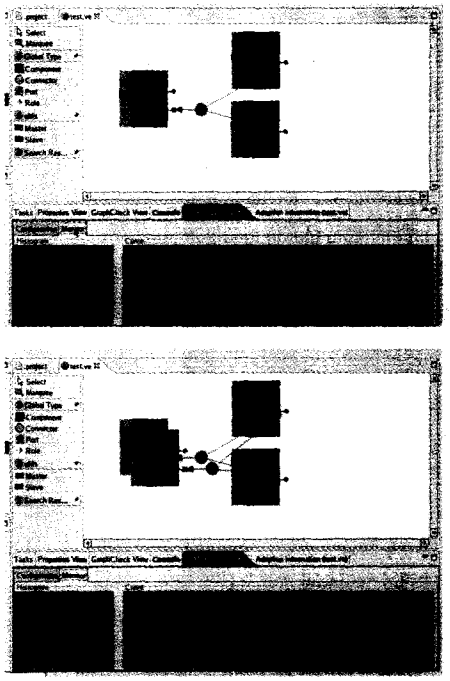

图6 自适应前的系统状态(上)和自适应后的系统状态(下)

图6中,每个图形的上半部分是当前系统体系结构的图形表示,下半部分是系统响应时间的柱状图和曲线图。曲线图中红色横线代表触发自适应调整的阈值,并且下图是上图的延续,下图的1、2时刻和上图的1、2时刻相同。初始模拟的客户数是200。我们在1时刻增加了200个客户,此时虽然系统响应时间延长了,但是还不足以触发自适应。我们在2时刻又增加了200个客户。此时系统响应时间超过阈值,于是触发自适应,增加一个新的 master 节点。新的 master 在4时刻开始正常工作,系统自适应调整完毕,系统响应时间恢复到一个用户可以接受的水平。体系结构图也由上图的单 master 转换成了下图的双 master。

从这个例子可以看出,Artemis-MAC 对基于本体的自适应软件开发支持良好,基于本体的软件自适应机制是有效的。

5 相关工作

我们的工作主要集中在以下3个方面:需求引入,决策要素建模,自适应机制重构。这一节将论述这3个方面的相关工作。

需求分析作为软件工程的第一步,对软件系统的成败起决定性的作用。因此学者在需求工程方面做了很多研究[17]。为了缩小问题空间和解空间之间的距离,让需求工程的研究成果更好的为软件开发服务,人们研究了需求到软件体系结构的转化[18,19]。然而,这种转化往往是单向的、一次性的,转化完成后,问题空间和解空间的联系就断了。目前已经有学者注意到了这个问题[5],并且做了在需求向软件体系结构转化过程中根据需求导出软件自适应流程的尝试。然而,在自适应软件中显式表达需求相关要素,并据此展开系统性的研

究的工作还未见。

现有软件自适应研究往往局限于解空间，因此对自适应决策要素的认识不够全面。决策要素的表达和操纵也大多采用程序设计语言或类程序设计语言的语言设施来实现，抽象层次低且通用性差。普适计算[20]中的情景建模在要素数据的建模和操纵上做了比较深入的研究。文[12,21]利用本体对普适计算环境下的情景建模，采用 OWL/RDF 作为本体描述语言。它们通过对基于 OWL/RDF 描述的情景进行推理得到一些高层的信息，诸如人的位置，当前行为等。它们的情景建模方法和推理机制对我们的工作有一定的启示。

在基于软件体系结构的软件自适应研究方面，CMU 的 Garlan 教授领导的 Rainbow 项目[3]具有一定的代表性。它采用"不变式－策略"的方式表达自适应逻辑。系统的约束条件用不变式表达。不变式不满足时，就根据与其对应的策略调整系统。调整策略采用编程语言的风格来描述。它的决策要素是根据需要选取的某些构件的属性，通过一组探测器来获取。它没有在决策要素认识和决策要素建模上作进一步的探讨。

本文的工作是我们前期工作的延续和深化。和已发表的工作[4,7,11]相比，本文途径的特点在于力图通过决策要素本体把问题空间的系统需求和解空间的软件自适应措施自然地联系起来。为此，本文对基于软件体系结构的自适应的决策要素做了深入的分析，研究了基于本体的决策要素建模方法，为自适应软件的决策要素表达和操纵提供了基础。利用 OWL/RDF 描述的决策要素更加标准化，通过组织良好的推理规则表达的自适应逻辑更加易于理解和管理。在此基础上，我们对原有的自适应机制做了调整，增加了自适应决策层，由它管理决策要素和自适应逻辑并做出自适应决策。调整后的自适应机制分层更加合理，进一步简化了自适应软件的开发，提高了自适应软件的可复用性和可维护性。同时，我们在 Artemis-MAC 系统中实现了基于本体的软件自适应机制，为自适应软件的开发提供了一个可用的支撑平台。

小结 为了更好地管理日益复杂、庞大的软件系统，人们提出了软件自适应，期望软件通过自我调整来确保其行为质量符合用户需求。我们分析了现有的基于软件体系结构的自适应机制，发现大多忽视了问题空间在软件自适应中的作用，以及相关的决策要素认识不完备，决策要素的表达和操纵缺乏通用性等问题。为此，我们提出在软件自适应中综合运用问题空间和解空间中的决策要素，采用本体对决策要素建模，从而为决策要素的统一表达和操纵提供基础。采用推理规则描述的自适应逻辑与程序代码分离，具有更高的抽象层次和更好的复用性。在此基础上，我们重新组织了自适应软件的结构，设计了一个基于本体的软件自适应机制。

为了验证基于本体的软件自适应机制的有效性，同时也为了支持开放环境下自适应软件的开发，我们在 Artemis-MAC 系统中实现了这一机制，并通过一个应用实例展示了自适应效果。

下一步，我们将在自适应软件决策要素的发现、组织和建模方法上做更深入的研究。同时，我们将进一步完善自适应决策层的架构，把它设计成一个开放的、支持插件式扩展的自适应管理和决策平台。

参 考 文 献

1 Yang Z, Cheng B H, Stirewalt R E K, et al. An Aspect-oriented Approach to Dynamic Adaptation. In: Proceedings of the first workshop on Self-healing systems, 2002

2 Reilly D, Taleb-Bendiab A, Laws A, et al. An Instrumentation and Control-based Approach for Distributed Application Management and Adaptation. In: Proceedings of the first workshop on Self-healing systems, 2002

3 Garlan D, Cheng Shang-Wen, Huang An-Cheng, et al. Rainbow: Architecture-based Self-adaptation with Reusable Infrastructure. IEEE Computer, 2004, 37(10): 46～54

4 许婷. 一种基于软件体系结构的自适应软件框架及其实现:[硕士论文]. 南京大学, 2005

5 Hawthorne M J, Perry D E. Exploiting Architectural Prescriptions for Self-Managing, Self-adaptive Systems: A Position Paper. In: WOSS'04, 2004

6 Gruber T R. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 1993, 5: 199～220

7 俞春，马骞，马晓星，等. 一种面向体系结构的软件系统自适应机制. 南京大学学报(自然科学)，2005，42(2)：120～130

8 http://www.w3.org/TR/owl-ref/

9 http://www.w3.org/RDF/

10 http://jena.sourceforge.net/

11 马晓星，余萍，陶先平，等. 一种面向服务的动态协同架构及其支撑平台. 计算机学报，2005(4)

12 Wang Xiao-Hang, Gu Tao, Zhang Da-Qing, et al. Ontology-based Context Modeling and Reasoning Using OWL. Pervasive Computing and Communications Workshops, 2004

13 van Lamsweerde A. Goal-oriented Requirements Engineering: A Guided Tour. In: The 5th IEEE International Symposium on Requirements Engineering, 2001

14 Garlan D, Monroe R, Wile D. Acme: An Architecture Description Interchange Language:[CMU Technical Report]. 1997

15 Monroe R T, Kompanek A, Melton R, et al. Architectural Styles, Design Patterns, and Objects. IEEE Software, January 1997

16 Zhou Yu, Pan Jian, Ma Xiaoxing, et al. Applying Ontology in Architecture-based Self-Management Applications. In: The 22nd Annual ACM Symposium on Applied Computing, Seoul, Korea, March, 2007

17 van Lamsweerde A. Requirements Engineering in the Year 00: A Research Perspective. In: ICSE 2000, June 2000. 5～19

18 Liu Lin, Yu E. From Requirements to Architectural Design - Using Goals and Scenarios. In: ICSE-2001 Workshop: From Software Requirements to Architectures (STRAW 2001) May 2001, Toronto, Canada

19 Brandozzi M, Perry D E. Transforming Goal-oriented Requirement Specifications into Architecture Prescriptions. In: ICSE-2001 Workshop: From Software Requirements to Architectures (STRAW 2001) May 2001, Toronto, Canada

20 Satyanarayanan M. Pervasive computing: vision and challenges. Personal Communications. IEEE, 2001, 8(4): 10～17

21 Chen H, Tim Finin. An Ontology for Context Aware Pervasive Computing Environments. The Knowledge Engineering Review, 2004