软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(1):17−27 [doi: 10.3724/SP.J.1001.2011.03813]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种支持软件可信演化的构件模型*

丁 博[1+], 王怀民[1,2], 史殿习[1], 李 骁[1]

[1](国防科学技术大学 计算机学院,湖南 长沙 410073)

[2](并行与分布处理国防科技重点实验室,湖南 长沙 410073)

Component Model Supporting Trustworthiness-Oriented Software Evolution

DING Bo[1+], WANG Huai-Min[1,2], SHI Dian-Xi[1], LI Xiao[1]

[1](School of Computer, National University of Defense Technology, Changsha 410073, China)

[2](National Defense Laboratory for Parallel and Distributed Processing, Changsha 410073, China)

+ Corresponding author: E-mail: dingbo@nudt.edu.cn

Ding B, Wang HM, Shi DX, Li X. Component model supporting trustworthiness-oriented software evolution. *Journal of Software*, 2011,22(1):17−27. http://www.jos.org.cn/1000-9825/3813.htm

Abstract: Environment-Driven adaptation is an important means ensuring software integrity. Confronted with scenarios not anticipated during the developmental stage, the predefined adaptability of the software should be adjusted to ensure that its behavior agree with the users' expectation. The premise of this kind of adjustment are efficient software engineering mechanisms. Based on the principles of the Separation of Concerns and the Dynamic Software Architecture (DSA) technology, this paper proposes a component model named ACOE (adaptive component model for open environment) that supports the online fine-grained adjustment to software adaptability. ACOE encapsulates adaptation concerns, such as sensing, decision, and execution into components, and connectors, and then supports their dynamic configuration with the DSA technology. As a result, a third-party can adjust the adaptability by selectively upgrading it when necessary. An ACOE container prototype and experimental applications are implemented to validate this approach.

Key words: trustworthy software; evolution; adaptive software; component model; dynamic software architecture

摘 要: 对环境的适应是软件保证其可信的重要手段.当应用场景超出开发阶段的预设时,软件的环境适应能力需要能够在线调整,以保证其行为和结果仍可符合用户预期.这一调整的前提是软件工程层面的高效支持机制.基于关注点分离原则和动态软件体系结构技术,提出了一种支持软件环境适应能力细粒度在线调整的构件模型ACOE(adaptive component model for open environment).ACOE 将软件环境适应能力中的感知、决策、执行等关注点封装为独立的构件和连接子,通过动态软件体系结构技术来支持它们的在线重配置,从而使第三方可在必要时通过有选择性的更新来调整适应能力.实现了支持 ACOE 构件模型的容器原型,并通过实验验证了其有效性.

关键词: 可信软件;演化;适应性软件;构件模型;动态软件体系结构

中图法分类号: TP311 **文献标识码**: A

---

* 基金项目: 国家自然科学基金(90818028); 国家重点基础研究发展计划(973)(2011CB302600); 国家杰出青年科学基金(60625203)

收稿时间: 2009-06-15; 修改时间: 2009-09-11; 定稿时间: 2009-12-07

软件可信是指软件行为符合用户预期、满足用户的需求[1].传统的形式化证明、测试等软件可信保证手段均在部署前实施,需要基于对未来运行环境的某些假设.然而,随着软件应用模式的变革,软件的环境不再是封闭静态的,它有可能超出开发阶段的预期[2].例如,对于具备长生命周期的超大规模软件系统[3]以及许多与物理世界紧耦合的分布式实时嵌入(distributed real-time embedded,简称 DRE)系统,很难在开发阶段即对其未来运行环境做出精确预测.

环境是动态变化的,软件需要具备适应能力;对于开发阶段未预期的环境及停机维护代价高昂的系统,这种环境适应能力更需要能够在线调整.我们称以可信为目标的软件演化为可信演化,而环境适应能力的在线调整是可信演化的重要内容.期望软件完全自主地实施这种演化尚不现实,它往往需要维护人员、其他软件实体等第三方的参与,本文工作仅仅关注如何在软件工程层面为适应能力在线调整提供使能机制.

软件的环境适应能力可细分为感知、决策及执行的能力[4]:在感知到环境发生变化后,软件做出相应决策,进而执行方法调用、参数改变或者结构调整等动作.在许多情况下,为了调整软件的适应能力,我们只需对上述过程作部分修改.一个例子是中间件层通信链路自适应切换:中间件底层同时支持多种通信链路,且内置了随链路可用性而透明切换的机制,但在某个非预设、需要高可靠通信的场景下还要求能够依据带宽来切换,此时只需添加带宽的感知手段和相应决策逻辑,而没有必要修改在各个链路上收发消息的代码.另一个例子是楼宇火警监测 DRE 系统:由于某个房间用途变更(如改为厨房),烟雾浓度与火警触发的关系不再遵循开发阶段的假设,此时只需修改对烟雾浓度进行处理的适应决策逻辑,烟雾浓度感知手段和报警业务流程都无需改变.

因此,要支持软件适应能力的高效在线调整,在软件工程层面有必要实现感知、决策和执行三者的关注点分离,并支持它们的在线重配置.针对这一挑战,我们提出了适应性构件模型 ACOE(adaptive component model for open environment).ACOE 引入感知构件和行为构件类型,在构件组装层面引入抽象“环境变化-适应动作”关系的策略连接子,从而使得感知、决策和执行这 3 个关注点可以独立表达和封装.在此基础上,ACOE 基于动态软件体系结构(dynamic software architecture,简称 DSA)技术[5]实现这些关注点的在线重配置.上述机制使得第三方可以通过对各个关注点的有选择性更新来细粒度地调整软件适应能力,在环境超出开发阶段的预期时保证软件的可信.

本文第 1 节给出 ACOE 构件模型的基本组成,概述其对软件适应能力在线调整的支持机制.第 2 节阐述 ACOE 构件模型中的构件类型和构件构造方法.第 3 节阐述 ACOE 构件模型的组装机制,关注在该构件模型中如何实施可信演化.第 4 节给出容器原型并通过实验验证本文工作.第 5 节是相关工作及比较.

1 ACOE 构件模型概述

适应是指软件感知环境变化并据此调整自身行为的过程[6].这一概念针对的是环境及其动态变化,并不强调环境是否可预期,因此文献[6]指出,适应性软件可以分为封闭和开放两大类:前者的所有适应动作在开发阶段即已指定,后者则可以实施非预设的适应动作.要使软件有能力实施非预设的适应动作,就需要对软件进行修改.ACOE 构件模型通过对软件适应过程中感知、决策、执行三者的关注点分离和在线重配置来支持上述修改的高效实施.

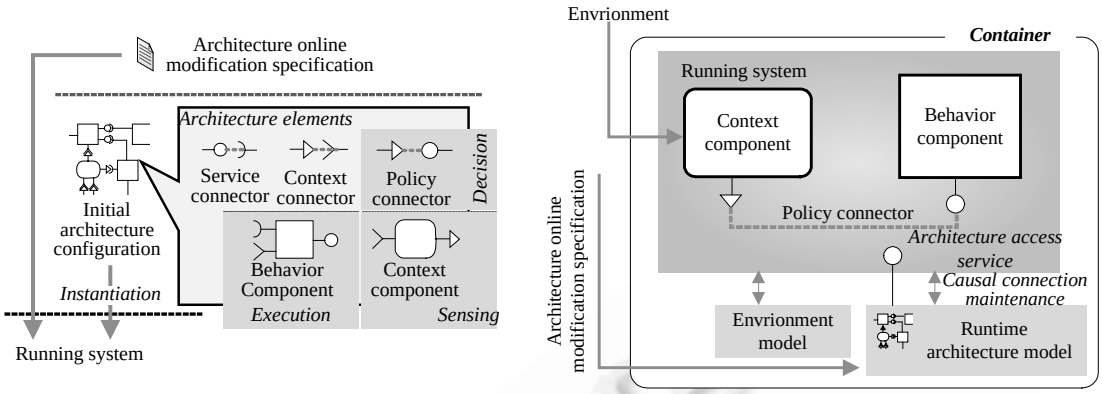
1.1 ACOE构件模型中的关注点分离机制

ACOE 定义了两种构件类型(如图 1(a)所示):行为构件和感知构件.感知构件是 ACOE 中对环境感知能力进行封装的手段,它负责捕获环境变化并输出相应的上下文(context)事件,以建立环境模型和触发适应动作,在功能上区别于对执行环节进行封装的普通行为构件.

在构件组装方面,ACOE 延续了软件体系结构的思想,亦即将系统抽象为由构件和连接子所组成的网络[5],组装因此表现为选择一组构件实例并通过适当的连接子将它们连接在一起的过程[7].ACOE 使用面向组装的体系结构描述语言 AcoeADL 来描绘组装后的系统蓝图.

在 ACOE 连接子中,策略连接子是最重要的一类连接子.它绑定感知构件和行为构件,抽象“环境变化-适应动作”关系,是封装软件适应过程中决策逻辑的手段.策略连接子的适应动作部分可以指定功能适应与结构适

应[8,9]:前者是指软件可以对其行为的语义进行调整,具体表现为在策略连接子中调用行为构件的方法;后者则是指软件能够对自身结构进行调整,如增删、替换构件和连接子等,具体表现为策略连接子中对容器所提供的体系结构修改服务的调用.



(a) Major conceptual entities

(a) 主要概念实体

(b) Adaptation and adaptability adjustment at runtime

(b) 运行时的适应和适应能力在线调整

Fig.1 An overview of ACOE component model

图 1 ACOE 构件模型概览

1.2 实现软件适应和适应能力在线调整

关注点已建模为构件和连接子,因此,关注点的在线重配置可以通过动态软件体系结构技术来实现.我们引入显式的运行时软件体系结构模型[10]来支持软件体系结构的在线变化.这一模型与实际系统因果关联[5],也就是说,它真实地反映了实际运行系统的当前状态,并且对该模型的任意修改均将被映射到实际系统之上.运行时软件体系结构模型可以带来如下一些直接的好处:(1) 提供实时的系统高层视图,为体系结构在线变化提供决策依据;(2) 暴露重要的系统属性和约束,从而有助于判断变化的正确性;(3) 提供对软件体系结构进行修改的统一入口,使体系结构变化可以独立于具体的功能代码.

在 ACOE 构件模型中,容器负责维护运行时软件体系结构模型及其与实际系统的因果关联.结合底层容器,ACOE 可以(如图 1(b)所示):

(1) 支持软件适应.通过获取感知构件的输出,容器的环境模型部件维护一个当前相关环境状态的镜像.在此基础上,容器在策略连接子指导下驱动软件适应过程:当策略连接子中指定的感知构件产生上下文事件,且当前环境状态符合一定条件时,指定的行为构件方法或体系结构修改操作将被执行;

(2) 支持软件适应能力的细粒度在线调整.第三方可以对运行时软件体系结构模型进行修改.AcoeADL 同时也充当体系结构修改语言的角色,它的一个子集可以用来单独指定对运行时体系结构的修改动作,如增删替换封装感知能力的感知构件、封装执行能力的行为构件、封装决策逻辑的策略连接子等.容器可以接受修改规约,修改运行时软件体系结构模型,进而通过因果关联作用到实际运行系统上.

仍以本文开始部分所给的两个例子为例,对于通信链路自适应切换的例子,只需增加新的带宽感知构件和封装了新的自适应切换逻辑的策略连接子,中间件的其他部分(如封装为行为构件的链路切换过程、消息收发代码等)不会受到影响;对于楼宇火警监测 DRE 系统,我们可以在线替换连接感知构件和报警行为构件之间的策略连接子来改变火警阈值.在相应的修改完成后,软件即可应对原来未预期的环境.上述调整过程是细粒度的,与之形成对比的是传统构件模型下的粗粒度演化——由于缺乏模型层的支持机制,感知代码及决策逻辑可能被混杂到相对稳定的业务逻辑之中,一旦环境发生变化,必须三者同时一起更新.

2 ACOE 构件类型和构造方法

文献[11]指出,构件模型是基于构件的软件开发方法的核心,它由构件语义、构件语法和构件组装 3 部分组成:构件语义说明了构件到底是什么;构件语法规定了构件是如何被定义、构造和表示的;构件组装则说明构件是如何被静态和动态组装的.我们将沿此线索阐述 ACOE 构件模型:在本节,说明 ACOE 构件类型以及构件如何被定义和构造;在下一节,介绍 ACOE 构件组装机制.

2.1 构件类型

对环境的显示表达是 ACOE 构件模型实现软件适应的基础.环境的表达可以通过上下文建模的方式来实现,且已经存在多种建模方法[12].ACOE 在构件模型层面使用已被广泛接受、易于管理的键值对模型.

定义 1(上下文及上下文事件). 上下文 c 可定义为二元组$\langle name,type\rangle$,其中,$name$ 是上下文的名称,$type$ 是上下文的类型.上下文 c 在任一确定的时刻均有一个类型为 $c.type$ 值 v;上下文事件 e 被定义为三元组$\langle c,v_1,v_2\rangle$,代表上下文 c 的值从 v_1 变化到 v_2,其中 $type(v_1)=type(v_2)=c.type$.

简单地说,上下文是环境某一个维度(如可用内存、温度、位置等),其定义可以来源于相应的上下文本体;上下文的值代表了环境在该维度的状态;上下文事件代表的则是该维度状态的变化.在上述定义的基础上,可以区分构件的上下文端口和服务接口:前者可以输出上下文事件,用来触发软件适应过程、更新容器内部的环境模型等;后者则是一般意义上包括方法、属性等的接口.

定义 2(上下文端口). 上下文端口 CP 可以在指定的上下文 c 发生变化时,输出相应的上下文事件.上下文的类型 $c.type$ 同时也被称为该上下文端口的类型.

感知构件是对环境感知手段的封装,提供一到多个上下文端口,同时可以依赖零到多个上下文端口,亦即可以实现基于构件的上下文聚合[12].行为构件则沿续了一般意义上构件的语义,它是对具体业务行为的封装,通过零到多个服务接口对外提供服务,并可以依赖其他接口.

定义 3(感知构件和行为构件). 感知构件是对环境感知手段进行封装的可组装软件单元,可定义为三元组$\langle S_{CP_p},S_{CP_r},Body\rangle$,其中,$S_{CP_p}$ 是所提供的上下文端口集合,S_{CP_r} 是所依赖的上下文端口集合,$Body$ 是构件的实现体.行为构件是对具体业务行为进行封装的可组装软件单元,可定义为四元组$\langle S_{I_p},S_{I_r}S_{CP_r},Body\rangle$,其中,$S_{I_p}$ 是所提供的服务接口集合,S_{I_r} 是所依赖的服务接口集合,S_{CP_r} 是所依赖的上下文端口集合,$Body$ 是构件的实现体.

2.2 构件构造方法

在已有的构件模型中,构件外部特征的定义方法可以分为两大类:直接使用具体的编程语言(如 EJB)或使用专门的构件定义语言(如 CCM,COM/DCOM).后者一方面有助于实现不同语言、不同平台间的互操作,更重要的是,它可以使构件类型信息显式化,从而有助于未来对体系结构的修改.因此,我们在 ACOE 中引入了专门的构件定义语言 UCDL(ubiquitous component definition language)来定义构件,它是后文将介绍的 AcoeADL 语言的一部分.

UCDL 是对 OMG IDL2[13]语言的扩展,主要的扩展内容包括:(1) 引入 component 关键字及相关语法来定义构件;(2) 引入 context 关键字及相关语法来定义上下文事件;(3) 引入 uses,provides,multiple 等关键字来修饰接口/端口;(4) 引入 state 关键字来定义构件可外部访问的状态;等等.在后文的图 4(a)中,我们给出了以第 4.2 节应用为背景的 UCDL 片断,它定义了行为构件 DisplayUnit、火警概率感知构件及相关数据结构.具体的 UCDL 语法可参见文献[14].

ACOE 构件的构造方法如后文的图 3 上部所示:专门设计的编译器可以将 UCDL 构件定义映射为 C++等具体编程语言下的构件框架,构件开发者可以填写构件实现并生成构件包.构件框架的接口中包括了普通服务、上下文事件处理和构件反射 3 类方法,其中:前二者由 UCDL 定义直接映射生成;反射方法则可以向容器报告构件类型、当前运行状态等基本元信息,也可以完成构件生命周期管理等操作.此外,构件实现框架还内建依赖代理,可以将本构件对其他构件的请求发给容器,由容器决定谁来具体完成该请求(如图 2 所示).

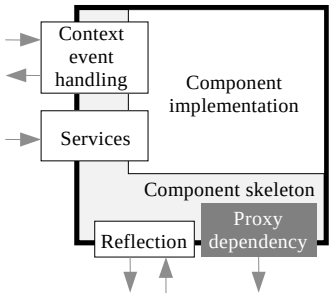


Fig.2 ACOE component skeleton

图 2 ACOE 构件框架

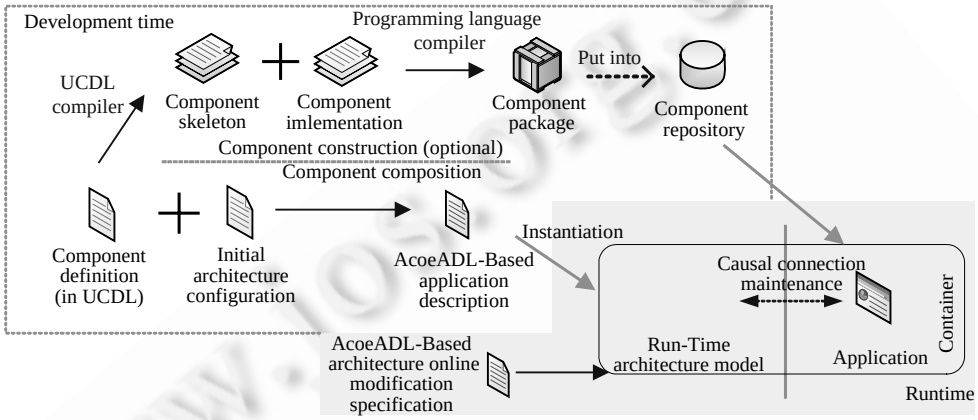


Fig.3 ACOE component construction and composition

图 3 ACOE 构件的构造和组装

3 ACOE 构件组装及可信演化的实施

组装是构件模型的核心[11].构件化方法通过可重用、相对稳定的构件和应用相关的组装来实现软件的高效构造.ACOE 构件模型从软件体系结构的角度实施构件的组装,即强调连接子的一阶实体地位,并使用体系结构描述语言来描述系统蓝图.本节首先给出 ACOE 构件模型中 3 种连接子的定义,然后介绍支持构件组装的 AcoeADL 语言以及如何定义体系结构修改规约来实施可信演化.

3.1 ACOE构件模型中的连接子

在软件体系结构中,连接子是与构件同等重要的一阶实体,它是对构件间交互机制的独立封装[15].在 ACOE 构件模型中,我们引入了 3 类连接子:负责服务接口绑定的服务连接子、负责建立上下文事件通道的上下文连接子以及负责封装适应决策逻辑的策略连接子.

定义 4(服务连接子). 服务连接子可以定义为一个四元组$\langle BComp_r,I_r,BComp_p,I_p\rangle$,其中:$BComp_r$ 是作为服务提供者的行为构件实例;$BComp_p$ 是作为服务消费者的行为构件实例;I_r 和 I_p 则具有相同的接口类型,

$$I_r \in BComp_r.S_{I_r}, I_p \in BComp_p.S_{I_p}.$$

定义 5(上下文连接子). 上下文连接子可以定义为一个四元组$\langle Comp_r,CP_r,CComp_p,CP_p\rangle$,其中:$Comp_r$ 是消费上下文事件的感知构件或行为构件实例;$CComp_p$ 是提供上下文事件的感知构件实例;CP_r 和 CP_p 具有相同的端口类型,$CP_r \in Comp_r.S_{CP_r}, CP_p \in CComp_p.S_{CP_p}$.

策略连接子负责封装适应的决策逻辑,绑定感知构件和行为构件.策略连接子的定义基于 ECA(event-

condition-action)[16]的模式,其具体实例如图 4(b)所示.

定义 6(策略连接子). 策略连接子可以定义为四元组$\langle CComp_p, CP_p, Conditions, Actions\rangle$,其中:$CComp_p$是提供上下文事件的感知构件实例,$CP_p \in CComp_p.S_{CP_p}$;*Conditions* 是加诸于环境模型上的条件;*Actions* 是方法调用序列,方法可以来自于行为构件实例提供的服务,也可以是容器所提供的体系结构修改服务.

在实际运行时,策略连接子由容器负责解释执行:容器监视所指定的上下文事件,依据所维护的环境模型检查条件是否满足,并在适当的时机执行行为构件的方法或者体系结构修改动作.

3.2 支持构件组装的AcoeADL语言

ACOE 构件组装包括选择所需的行为构件和感知构件,并通过服务连接子、上下文连接子和策略连接子绑定这些构件.其中,感知构件和策略连接子的选择由软件适应的具体需求决定.

在基于软件体系结构的组装方面,较有代表性的工作是文献[17]所提出的基于软件体系结构描述语言和中间件的组装方法.该方法使用面向组装的 ABC/ADL 语言来描述系统蓝图,使用中间件作为构件组装的运行时支撑设施.我们将这一方法应用到了 ACOE 构件模型之中,使用 AcoeADL 软件体系结构描述语言描述由构件和连接子组成的系统蓝图,使用容器来实例化该蓝图,通过软件体系结构的在线修改来实现可信演化.

软件体系结构描述语言是对软件系统概念架构的描述,它必须能够支持体系结构中构件、连接子和配置 3 个基本模块的显式规约[18].在 AcoeADL 中,构件规约由第 2.2 节介绍的 UCDL 语言描述;连接子规约是对定义 4 到定义 6 的细化,并直接嵌入在体系结构初始配置中(亦即不区分连接子与连接子实例);体系结构初始配置由一组构件实例和连接子组成,它们形成一个描述目标软件系统体系结构的图.图 4(b)所给配置描述示例指定了 3 个构件实例:*FireDisplayUnit*,*CA*1 和 *CA*2,以及一个策略连接子 AlarmActivation.

```
struct RawData{…}; //IDL2-described data structure
interface FireDisplay{…}; //IDL2-described interface
interface AlarmLamp{…};
…
component DisplayUnit {//Behavior component
    provides FireDisplay fd;
    uses AlarmLamp al;
    state bool InAlarm;
};
component ContextAggregation {//Context component
    provides context int FirePossibility;
    uses context RawData SmokeDensity;
    uses context RawData Temperature;
};
…
```

(a) Component definition
(a) 构件定义

```
configuration FireAlarm {
    component_instance DisplayUnit FireDisplayUnit;
    component_instance ContextAggregation CA1;
    …
    policy_connector AlarmActivation
        event CA1.FirePossibility;
        condition
    ((CA1.FirePossibility>60)&&(!FireDisplayUnit.InAlarm));
        action {
            FireDisplayUnit.FireDisplay.Alarm(true);
        }
    …
    constraint c1.ConnectorCount("CA","Temperature")≤10;
    constraint c2.InstanceCount("DisplayUnit")=1;
}
```

(b) Initial architecture configuration
(b) 初始体系结构配置

```
target_configuration FireAlarm;
policy_connector AlarmActivationNew //Policy Connector for the new environment
…
action{Container.AAS.ReplaceConnector("AlarmActivation","AlarmActivationNew");} //Modifying the architecture model
```

(c) Architecture online modification specification
(c) 体系结构在线修改规约

Fig.4 AcoeADL-Based architecture descriptions
图 4 基于 AcoeADL 的体系结构描述

3.3 使用AcoeADL语言实施可信演化

体系结构修改规约被用于第三方在运行时单独指定对软件体系结构的修改动作,实现基于动态组装的软件可信演化.体系结构修改规约通过 AcoeADL 的一个子集来指定——策略连接子 action 部分的语法被抽取出

来作为体系结构修改语言,第三方(人或软件实体)可以使用该语言书写修改规约,调用容器所提供的体系结构模型修改接口来实现构件和连接子的增删替换.图 4(c)即为一个实现策略连接子替换的体系结构修改规约,其中,*Container.AAS* 是容器所提供的体系结构访问和修改服务.

软件体系结构的在线修改可能会破坏其完整性[6].ACOE 通过缺省约束和用户自定义约束来维护这种完整性.缺省约束内建于 ACOE 构件模型之中,例如服务连接子只能绑定行为构件、连接子必须连接有效构件等.用户自定义约束则由应用开发者通过体系结构初始配置中的约束字段来指定,例如图 4(b)中的第 1 个约束指明了与 *CA*1.*Temperature* 端口关联的上下文连接子不能超过 10 个,第 2 个约束指明了在 *FireAlarm* 中不能动态实例化 *DisplayUnit* 类型的构件.在修改软件体系结构模型时,ACOE 容器会首先检查相关约束,只有在不违反约束时才会执行修改动作.

4 原型验证及实验

本节将给出我们所实现的 ACOE 容器原型,并结合在其上所开发的实例来验证 ACOE 构件模型的有效性.

4.1 支持ACOE构件模型的容器原型

除了扮演传统的构件运行环境的角色外,ACOE 容器也是实现软件适应和适应能力在线调整的基础设施.我们在分布对象中间件 StarBus[19]的基础上实现了一个 ACOE 容器原型 ACContainer,其主要部件如图 5 所示.其中:构件生命周期管理部件负责完成构件实例化、激活、卸载等生命周期操作;事件服务提供分布式事件发布\订阅服务,是上下文连接子的实现部件;服务通道部件负责依据服务连接子转发服务接口的访问请求;适应引擎负责解释执行策略连接子;环境模型保存容器中当前构件和连接子所关心的上下文的最新值,它是依据感知构件所输出的上下文事件维护的;体系结构模型本地视图保存运行时软件体系结构模型,其内容包括构件、连接子的描述以及它们的重要状态(如所处生命周期阶段)等.

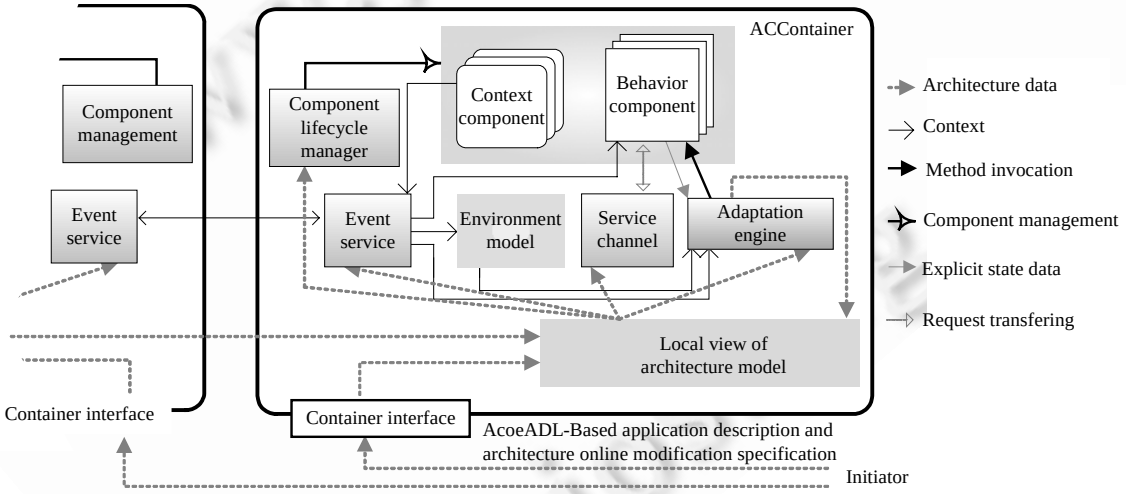


Fig.5 Container prototype supporting ACOE component model

图 5 支持 ACOE 构件模型的容器原型

ACContainer 采用与 *K*-Component 类似的分布式软件体系结构模型[20],即每个节点上仅维护与本地构件和连接子有关的体系结构视图.这一设计的目的是通过非集中式管理来获取可伸缩性,但也引入了体系结构分发、同步等问题.为了支持软件体系结构模型到多个节点的映射,我们引入了发起者(initiator)这一概念.发起者负责读入要实例化的软件体系结构模型,依据部署规划进行分析和切割,然后将相关的体系结构模型和构件包等推送到目标节点上.文献[21]对发起者及部署规划机制等作了进一步说明.

ACContainer 在运行时维护两个方面的体系结构一致性[6]:体系结构模型与实际运行系统之间的因果关联;

系统内部状态的一致性,如构件替换时的状态迁移等.前者通过反射机制实现:当体系结构模型发生变化时,构件生命周期管理部件负责映射构件相关的变化,服务通道和事件服务分别负责服务连接子和上下文连接子相关的变化,适应引擎则负责策略连接子相关的变化.当实际系统体系结构发生变化时(如构件意外失效),上述部件通过轮询构件的反射接口获知变化,相应的状态会被反馈到体系结构模型中.对于系统内部状态的一致性,ACContainer 通过模型修改动作的原子性保证、第 3.3 节中完整性维护机制、调用构件显式实现的状态保存/恢复接口来提供有限的支持.

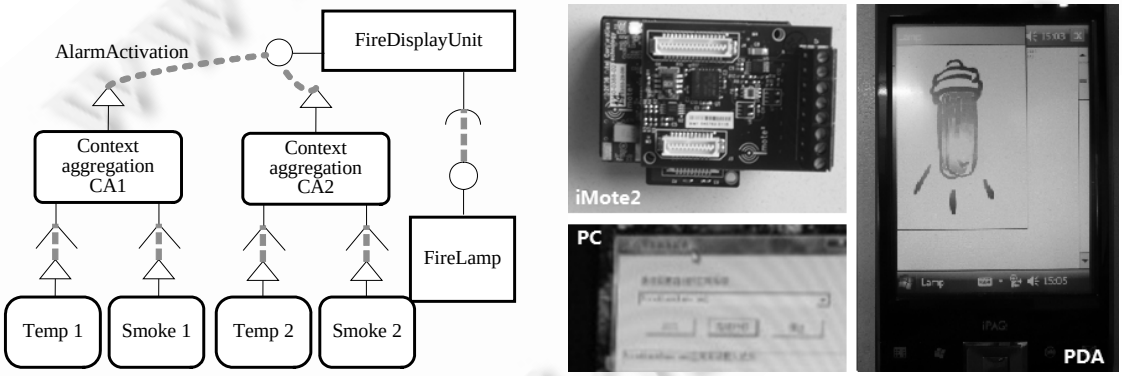
在上述设计的基础上,ACContainer 可以支持适应和适应能力的在线调整.适应引擎负责依据策略连接子驱动软件适应过程:事件服务将上下文事件推给适应引擎,适应引擎从环境模型读取当前环境状态,检查指定的条件是否满足,执行体系结构模型修改动作或是调用具体的构件方法.对于软件适应能力的在线调整,第三方在确定调整的时机和内容后,书写如图 4(c)所示形式的体系结构修改规约.ACContainer 的模型修改接口可以接受规约,修改体系结构模型本地视图,并通过前述一致性维护机制映射到实际运行系统上.

ACContainer 目前支持基于 Windows/Linux 的 PC 机、基于 Windows Mobile 的手持设备和基于 ARM-Linux 的 iMote2 传感器,并提供了支持 C++语言的 UCDL 编译器等开发工具.

4.2 实例分析

我们在 ACContainer 之上实现了本文开始部分中提到的楼宇火警监控 DRE 模拟应用.该应用具体背景如下:每个房间内部署有多个烟雾和温度传感器,它们所收集的环境状态信息送到本房间的上下文汇聚点;上下文汇聚点负责聚合出火警概率并发送给楼宇中央火警显示仪,如果任何一个房间发生火警,则开启火警灯.

该应用的部分初始体系结构配置如图 6(a)所示(以两个房间为例),其中温度、烟雾和上下文聚合等感知构件封装感知能力,AlarmActivation 策略连接子封装决策能力,火警显示、火警灯控制等行为构件封装执行能力,从而实现了软件适应能力中感知、决策和执行的关注点分离,进而可以通过 ACContainer 的接口来实现它们的在线重配置.应用由一组 iMote2 传感器(部分同时充当汇聚点)、充当火警显示仪的 PC 机及模拟火警灯的 PDA 组成.图 4 已给出该应用部分的 AcoeADL 描述,图 6(b)给出了参与设备及实际运行界面.



(a) Initial architecture configuration
(a) 初始体系结构配置

(b) Participant devices and snapshots
(b) 参与设备及其运行界面

Fig.6 Simulated fire alarm application based on ACOE
图 6 基于 ACOE 构件模型的火警监控模拟应用

我们通过如下场景来验证 ACOE 构件模型对软件适应和适应能力在线调整的支持:

(1) 软件适应.上下文聚合构件与火警显示构件由图 4(b)中的 AlarmActivation 策略连接子连接,当上下文聚合构件所输出的火警概率超过策略连接子中条件部分的阈值时,火警显示仪上显示火警并开启火警灯.

(2) 软件适应能力的细粒度在线调整.由于未预期的原因(如房间用途变更等)导致火警频繁误报,火警监控系统在新的环境下已无法正常工作.我们通过图 4(c)中的体系结构修改规约实施上下文 AlarmActivation 策略

连接子的在线替换,适当提高新连接子中火警的阈值,从而使火警监控系统具备了应对新环境的能力.

4.3 性能评估

策略连接子是 ACOE 实现关注点分离的手段之一,我们对引入策略连接子所带来的性能损失进行了测试.在楼宇火警监控模拟应用中设定了两个场景:场景 1 中,上下文聚合构件与火警显示构件之间通过 AlarmActivation 策略连接子连接,策略连接子、火警显示构件位于同一容器中;场景 2 中,上下文聚合构件与火警显示构件直接使用上下文连接子连接,适应的决策逻辑被硬编码到火警显示构件中.将火警概率的上下文事件抵达容器到其在同一容器内触发(或不触发)火警显示作为一个处理周期,通过比较两种场景下的周期时间,可以确定适应决策逻辑动态解释和硬编码两种方式的性能差异.

我们分别在 Windows 和 Windows Mobile 平台下进行了测试.火警显示构件的实现在不同平台下略有差异.输入的火警概率分为可触发火警响应(*A*)和不可触发火警响应(*B*)两种.每个周期涉及到上下文事件分发、策略执行、构件调用等多个动作,对每种情况作 100 次测试后取周期时间的平均值,测试结果如图 7 所示,表明策略连接子以较小的性能代价换来了适应决策逻辑单独在线维护和更新的可能性.

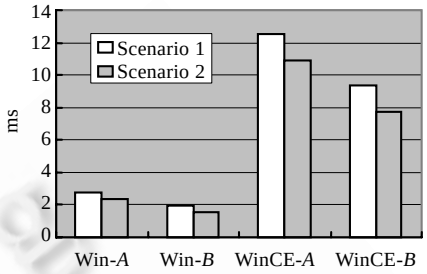


Fig.7 Performance loss of policy connector

图 7 策略连接子的性能损失

5 相关研究工作

软件适应领域的研究者已经认识到关注点分离原则在软件适应能力调整中的重要性.较为常见的实现方法是使用策略、规则或其他手段来单独描述软件适应的决策逻辑,然后通过对决策逻辑的更新来调整适应能力.文献[22]给出了一个实现软件自适应的规则模型;Chisel[23]面向完全不可预期的动态变化,使用策略来描述适应逻辑,通过对策略的动态更新来为软件添加新的适应能力;*K*-Component 通过适应契约描述语言(ACDL)来描述适应逻辑,实现体系结构在线变化过程中的关注点分离[24];MADAM 区分功能实现和适应性行为使能两个关注点,通过构件的附加属性、效用函数等来支持软件的适应动作[25].上述方法均只强调适应决策逻辑从软件功能实现中分离,而本文强调感知、决策、执行三者的关注点分离和在线重配置,如本文开始部分所分析,感知能力的独立表达和在线重配置对于软件适应能力的调整是十分必要的.此外,上述方法大多仅强调关注点分离,本文则进一步给出了基于构件模型和 DSA 技术来实现关注点在线重配置的方法.

与本文工作密切相关的另一方面是基于体系结构的软件自适应.文献[6]于早期提出了一种基于体系结构的软件自适应模型,由描述适应过程的适应管理和实现体系结构在线变化的演化管理两部分组成.除了前面已提及的 *K*-Component,MADAM 等项目外,Rainbow 在体系结构的在线修改过程中强调软件重用[26];FORMAware 强调对体系结构重配置时功能的完整性和体系结构约束不被违反[27];Fractal 构件模型及其框架在设计时即考虑体系结构配置的变化[28];等等.上述工作关注于基于软件体系结构模型实现适应,这些适应动作是在开发阶段通过 ACDL(*K*-Component),Strategy(Rainbow)等来预定义的,而本文工作则关注如何在运行时实现软件适应能力的在线调整.例如,通过增加策略连接子来引入新的适应动作、替换感知构件来获取新的上下文等.

网构软件[29]运行平台基于软件体系结构反射支持结构适应,通过自主构件来实现构件内部从感知到行为执行的功能适应[30].文献[31]提出了一种以“环境信息显式化、互动方式层次化、体系结构可演化”为特征的环境驱动模型.本文工作以上述体系结构演化和环境显式维护的思想为基础,针对非预设环境下软件适应能力调整这一挑战,提出了支持软件适应能力中感知、决策、执行三者关注点分离和在线重配置的构件模型.

6 结束语

运行时的适应和调整已成为保证软件可信的重要手段.本文以软件环境可能超出开发阶段预期为背景,提

出了一种支持软件适应能力细粒度在线调整的适应性构件模型 ACOE.ACOE 将软件适应能力中感知、决策和执行三者建模为独立的软件体系结构元素,并通过对运行时体系结构模型的修改来对上述关注点进行在线重配置,从而使得软件环境适应能力中各个维度均可以在运行时刻被第三方有选择性、细粒度地修改,软件能够在新环境下表现出符合用户预期的行为.本文工作已在支持 ACOE 构件模型的容器原型及其上应用中得到验证,后续工作包括上下文表达能力的扩展、严格的体系结构完整性维护机制、ACConainter 容器的完善、体系结构在线调整效果的评估等.

References:

[1] Wang HM, Tang YB, Yin G, Li L. Trustworthiness of Internet-based software. Science in China (Series E), 2006,36(10): 1156−1169 (in Chinese with English abstract).

[2] Baresi L. Toward open-world software: Issue and challenges. Computer, 2006,39(10):36−43. [doi: 10.1109/MC.2006.362]

[3] Northrop L, Feiler PH, Pollak B, Pipitone D. Ultra-Large-Scale Systems: The Software Challenge of the Future. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2006.

[4] Salehie M, Tahvildari L. Self-Adaptive software: Landscape and research challenges. ACM Trans. on Autonomous and Adaptive Systems, 2009,4(2):1−42. http://doi.acm.org/10.1145/1516533.1516538

[5] Mei H, Shen JR. Progress of research on software architecture. Journal of Software, 2006,17(6):1257−1275 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

[6] Oreizy P, Gorlick MM, Taylor RN, Heimbigner D, Johnson G, Medvidovic N, *et al*. An architecture-based approach to self-adaptive software. IEEE Intelligent Systems, 1999,14(3):54−62. [doi: 10.1109/5254.769885]

[7] Gruntz D, Murer S, Szyperski C. Component Software: Beyond Object-Oriented Programming. Massachusetts: Addison-Wesley, 2002.

[8] Dragan S. Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability: Adaptive Technologies and Applications. IGI Global, 2009. 26−58.

[9] Raibulet C. Facets of adaptivity. In: Proc. of the European Conf. on Software Architecture. 2008. 342−345. http://portal.acm.org/citation.cfm?id=1434586 [doi: 10.1007/978-3-540-88030-1_33]

[10] Wang QX, Huang G, Shen JR, Yang FQ. Runtime software architecture based software online evolution. In: Proc. of the Int'l Computer Software and Applications Conf. 2003. 230−235. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1245346 [doi: 10.1109/CMPSAC.2003.1245346]

[11] Lau KK, Wang Z. Software component models. IEEE Trans. on Software Engineering, 2007,33(10):709−724. [doi: 10.1109/TSE.2007.70726]

[12] Strang T, Linnhoff-Popien C. A context modeling survey. In: Proc. of the Workshop on Advanced Context Modelling, Reasoning and Management. 2004. http://www.citeulike.org/user/onierstrasz/article/3273347

[13] Yang Z, Duddy K. CORBA: A platform for distributed object computing. ACM SIGOPS Operating Systems Review, 1996,30(2): 4−31.

[14] Wu YL, Ding B, Shi DX, Liu H. Research and implementation of mapping mechanism of component model in ubiquitous computing environment. In: Proc. of the Joint Conf. on Harmonious Human Machine Environment. Wuhan, 2008 (in Chinese with English abstract). http://d.g.wanfangdata.com.cn/Conference_7014482.aspx

[15] Mehta NR, Medvidovic N, Phadke S. Towards a taxonomy of software connectors. In: Proc. of the Int'l Conf. on Software Engineering. 2000. http://portal.acm.org/citation.cfm?id=337201

[16] Mccarthy D, Dayal U. The architecture of an active database management system. ACM SIGMOD Record, 1989,18(2):215−224. [doi: 10.1145/66926.66946]

[17] Mei H, Chang JC, Yang FQ. Software component composition based on ADL and middleware. Science in China (Series F): Information Sciences, 2001,44(2):136−151. [doi: 10.1007/BF02713972]

[18] Medvidovic N, Taylor RN. A classification and comparison framework for software architecture description languages. IEEE Trans. on Software Engineering, 2000,26(1):70−93. [doi: 10.1109/32.825767]

[19] Wang HM, Wang YF, Tang YB. StarBus+: Distributed object middleware practice for Internet computing. Journal of Computer Science and Technology, 2005,20(4):542−551. [doi: 10.1007/s11390-005-0542-y]

[20] Dowling J. The decentralised coordination of self-adaptive components for autonomic distributed systems [Ph.D. Thesis]. Dublin: Trinity College, University of Dublin, 2004.

[21] Ding B, Shi DX, Wang HM. An adaptive software architecture for pervasive computing. In: Proc. of the Joint Conf. on Harmonious

Human Machine Environment. Wuhan, 2008 (in Chinese with English abstract). http://d.g.wanfangdata.com.cn/Conference_7014439.aspx

[22] Wang QX. Towards a rule model for self-adaptive software. SIGSOFT Software Engineering Notes, 2005,30(1):1–5. [doi: 10.1145/1039174.1039198]

[23] Keeney J. Completely unanticipated dynamic adaptation of software [Ph.D. Thesis]. Dublin: Trinity College, University of Dublin, 2004.

[24] Dowling J, Cahill V. The K-component architecture meta-model for self-adaptive software. In: Proc. of the Int'l Conf. on Metalevel Architectures and Separation of Crosscutting Concerns. 2001. 81–88. http://www.springerlink.com/content/yvuya4yb65aktft6/

[25] Paspallis N, Papadopoulos GA. An approach for developing adaptive, mobile applications with separation of concerns. In: Proc. of the 30th Annual Int'l Computer Software and Applications Conf. (COMPSAC). 2006. 299–306. http://portal.acm.org/citation.cfm?id=1170027

[26] Garlan D, Cheng SW, Huang AC, Schmerl B, Steenkiste P. Rainbow: Architecture-Based self-adaptation with reusable infrastructure. IEEE Computer, 2004,37(10):46–54. [doi: 10.1109/MC.2004.175]

[27] Moreira R, Blair G, Carrapatoso E. FORMAware: Framework of reflective components for managing architecture adaptation. In: Proc. of the 3rd Int'l Workshop on Software Engineering and Middleware. Orlando, 2002. http://www.springerlink.com/content/722578826r681411/ [doi: 10.1007/3-540-38093-0_8]

[28] Bruneton E, Coupaye T, Stefani JB. Recursive and dynamic software composition with sharing. In: Proc. of the 7th Int'l Workshop on Component-Oriented Programming (WCOP 2002). 2002. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.4559

[29] Yang FQ, Lv J, Mei H. Technical framework for Internetware: An architecture centric approach. Science in China (Series E), 2008, 38(6):818–828 (in Chinese with English abstract). [doi: 10.1007/s11432-008-0051-z]

[30] Mei H, Huang G, Zhao HY, Jiao WP. A software architecture centric engineering approach for Internetware. Science in China (Series E), 2006,36(10):1100–1126 (in Chinese with English abstract). [doi: 10.1007/s11432-006-2027-1]

[31] Lü J, Ma XX, Tao XP, Cao C, Huang Y, Yu P. On environment-driven software model for Internetware. Science in China (Series E), 2008,38(6):864–900 (in Chinese with English abstract).

附中文参考文献:

[1] 王怀民,唐扬斌,尹刚,李磊.互联网软件的可信机理.中国科学(E 辑),2006,36(10):1156–1169.

[5] 梅宏,申峻嵘.软件体系结构研究进展.软件学报,2006,17(6):1257–1275. http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

[14] 吴元立,丁博,史殿习,刘惠.普适计算环境下的构件模型映射机制的研究与实现.见:第 4 届和谐人机环境联合学术会议.武汉,2008. http://d.g.wanfangdata.com.cn/Conference_7014482.aspx

[21] 丁博,史殿习,王怀民.一种面向普适计算的适应性软件体系结构.见:第 4 届和谐人机环境联合学术会议.武汉,2008. http://d.g.wanfangdata.com.cn/Conference_7014439.aspx

[29] 杨芙清,吕建,梅宏.网构软件技术体系:一种以体系结构为中心的途径.中国科学(E 辑),2008,38(6):818–828. [doi: 10.1007/s11432-008-0051-z]

[30] 梅宏,黄罡,赵海燕,焦文品.一种以软件体系结构为中心的网构软件开发方法.中国科学(E 辑),2006,36(10):1100–1126. [doi: 10.1007/s11432-006-2027-1]

[31] 吕建,马晓星,陶先平,曹春,黄宇,余萍.面向网构软件的环境驱动模型与支撑技术研究.中国科学(E 辑),2008,38(6):864–900.

丁博(1978－),男,湖南攸县人,博士生,CCF 学生会员,主要研究领域为分布计算,自适应软件.

史殿习(1966－),男,博士,副教授,CCF 会员,主要研究领域为分布计算,普适计算.

王怀民(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布计算,网络安全.

李骁(1984－),男,博士生,主要研究领域为分布计算,中间件.