ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.7, July 2007, pp.1705−1714
DOI: 10.1360/jos181705

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

门户环境中基于语义数据协作应用集成方法*

宋靖宇+, 魏 峻, 万淑超

(中国科学院 软件研究所 软件工程技术研究开发中心,北京 100080)

A Semantic Data Coordination Based Approach for Application Integration in Portals

SONG Jing-Yu+, WEI Jun, WAN Shu-Chao

(Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+Corresponding author: Phn: +86-10-62661581, Fax: +86-10-62661580, E-mail: songjy@otcaix.iscas.ac.cn, http://www.ios.ac.cn

Song JY, Wei J, Wan SC. A semantic data coordination based approach for application integration in portals. *Journal of Software*, 2007,18(7):1705−1714. http://www.jos.org.cn/1000-9825/18/1705.htm

Abstract: At present, the significance of portal applications stems not only from being a handy way to access data but also from being the means of facilitating the integration with applications. The further integration of applications in portal context is actually the problem of Portlet interoperation. All the existing approaches for Portlet interoperation exhibit some drawbacks in sharing scope, standardization and difficulty to integrate legacy Web based applications. This paper proposes a novel semantic data coordination based approach to achieve Portlet interoperation. The proposed approach abstracts the Portlets involved in a Portlet interoperation to ShadowComponents, and then constructs semantic data associations between such ShadowComponents, and achieves Portlet interoperation based on ECA (event-condition-action) rules. Experimental results show that this proposed approach can achieve application integration in portal without modifications to the applications involved in an interoperation.

Key words: portal; Portlet interoperation; EAI (enterprise application integration); semantic coordination; ontology; event-condition-action rule

摘 要: 目前,门户的功能定位已经从传统的信息集成转向应用集成.门户环境中应用间的进一步集成实际上表现为 Portlet 互操作问题. 现有 Portlet 互操作方法在共享范围、标准兼容方面存在不足且难于集成已有应用系统.提出了一种基于语义数据协作的 Portlet 互操作方法,其基本思想是:将参与互操作的 Portlet 抽象为 ShadowComponent 组件,然后基于本体建立 ShadowComponent 之间的语义数据关联,并使用 ECA(event-condition-action)规则实现 Portlet 互操作.实验证明,该方法无须对已有应用作任何修改即可完成门户中的应用集成.

关键词: 门户;Portlet 互操作;企业应用集成;语义协作;本体;ECA 规则
中图法分类号: TP393 文献标识码: A

* Supported by the National Basic Research Program of China under Grant No.2002CB312005 (国家重点基础研究发展计划(973)); the national High-Tech Research and Development Plan of China under Grant Nos.2006AA01Z19B, 2006AA01Z161 (国家高技术研究发展计划(863))

Received 2005-12-15; Accepted 2006-05-08

门户通过集成异构应用、服务和数据资源,使用户实现了对信息的有效访问[1].早期门户,如 Yahoo!等主要采用分类、搜索等技术,目的是为了获得对 Internet 资源的有效访问.随着门户技术的不断发展,其使用场景发生了变化,门户在企业中获得了更为广泛的应用.这种变化使得门户更加面向应用集成.目前,门户的能力已经不仅体现在能够访问各种数据方面,而是更多地体现在它提供了一致、方便、有效的方法用于集成企业现有的多种应用系统方面,如 ERP(enterprise resource planning),CRM 及其他基于 Web 的信息系统等.

Java Portlet Specification(JSR168)[2]定义了应用和服务在表示层集成的标准,从而提供了基于组件的门户的技术基础.在本文中,若非特别指定,我们将简单地使用"门户"表示符合 JSR168 规范的门户.Portlet 是提供交互能力的 Web 小程序.通常情况下,门户会对 Portlet 产生的 HTML 标记片段添加标题和若干控制按钮,如最小化、最大化、编辑和删除等,形成 Portlet 窗口.然后,所有的 Portlet 窗口由门户组装为一个完整的门户页面.即使是这种简单的组合也具有一定的价值,因为所有的相关应用都同时展现在同一个页面中,用户可以更为快捷地获得更为全面的信息.但是,进一步的集成能力显然是必要的.一个 Portlet 显示的信息可能是另一个 Portlet 所需要的,这时,用户不得不手工复制或者在目标 Portlet 键入这些数据,从而影响了集成的连续性,并导致了一些不必要错误的发生.这说明 Portlet 之间需要某种互操作能力.此外,由于 Portlet 是唯一可被门户接受的组件类型,所以在门户需要集成已有应用时,这些应用必须首先以某种方式转换为 Portlet,然后才能在门户中使用.因此,在门户环境中,这些应用之间的进一步集成体现为它们所对应的 Portlet 的集成.从某种意义上讲,特别是在表示层次,门户环境下的应用集成问题实际上是 Portlet 之间的互操作问题.这进一步说明门户需要有效的 Portlet 互操作支持能力.

本文提出了一种基于语义数据协作的 Portlet 互操作方法.我们采用基于前端的实现策略,即主要通过扩展 Portlet 生成的标记片段及在原应用外围的包装来获得 Portlet 间的互操作能力.采用此方法不需要对集成的应用作任何修改.该方法综合使用了数据驱动的协作、信息提取及本体技术,其要点如下:

(1) 基于信息提取技术,为参与互操作的 Portlet 生成称为 ShadowComponent 的组件.ShadowComponent 由门户系统管理,并与其对应的 Portlet 保持状态同步.

(2) 将 ShadownComponent 的输入/输出数据映射到门户本体.由于每个 Portlet 的输入/输出数据具有不同的类型及语义,因此,基于本体完成数据之间的语义映射是实现互操作的必需环节.门户本体实现了不同 ShadowComponent 之间的数据基于语义的关联.

(3) 建立基于本体的协作机制实现语义数据驱动的 Portlet 互操作.ECA(event-condition-action)规则定义了数据在何时、以何种方式在 ShadowComponent 之间传递.

本文第 1 节给出 Portlet 互操作的概述及有关背景知识,并定义了为实现已有应用重用而对 Portlet 互操作提出的基本需求.第 2 节给出基于语义数据协作的 Portlet 互操作模型.第 3 节讨论互操作本体及如何基于本体实现语义数据关联.第 4 节分析基于语义数据协作的 Portlet 互操作实现的具体技术细节.第 5 节讨论相关工作.最后总结全文并给出下一步的工作方向.

1 背景与需求

1.1 使用场景

为了更好地描述 Portlet 互操作需求并说明我们的方法,本文使用如下场景作为用例.考虑一个汽车集团的市场部,部署了 3 个应用:订单管理系统(order management system,简称 OMS)、客户关系管理系统(customer relationship management system,简称 CRM)和业务智能系统(business intelligence system,简称 BI).这些系统分别包装为 OMPortlet,CRMPortlet 和 BIPortlet,并可以在门户中使用.为了解汽车的销售情况并挖掘潜在的市场,市场部经理构建了一个包含上述 3 个 Portlet 的门户页面.为进行市场分析,市场经理必须同每个 Portlet 交互并手动复制或输入数据.比如,要查看某个订单的客户详细资料,市场经理需要在 OMPortlet 中将选定的订单条目的 CustomerID 复制到 CRMPortlet,并提交查询请求获得有关信息.为了进一步了解该月该订单对应型号汽车在与该客户相同职业的客户群体中的销售情况,则需要将 OMPortlet 中的 ProductID 和 Date 以及 CRMPortlet 中获得

的 Occupation 信息复制到 BIPortlet 中.完成提交后,可以获得所需的分析数据.图 1 描述了上述过程.可以看出,整个过程非常繁琐而且容易出错,同时,严重影响了整个分析过程的流畅性.需要指出的更为重要的问题是,由于 3 个系统原来是完全独立开发并部署的,它们的数据并不完全匹配.具体来说,上述过程中存在 3 个数据匹配问题:① OMS 系统与 CRM 系统的 CustomerID 是不同的,必须通过其他系统或工具将 OMS 中的 CustomerID 转换为 CRM 中的 CustomerID,才能在 CRM 系统中找到指定客户;② OMS 中的 Date 采用 ISO8601 规范中的格式,而 BI 系统中使用的是私有格式,而且不需要日期信息;③ CRM 与 BI 系统对职业的分类方法不同,同样需要进行映射才能使用 BI 系统.

Fig.1 A scenario of Portlet interoperation

图 1 Portlet 互操作示例场景

Portlet 互操作需要提供的就是使上述过程更加流畅的机制,其关键是支持数据在各 Portlet 之间的关联与传递.举例来说,市场经理希望的使用过程是:通过在 OMPortlet 中选定某个订单并点击特定的按钮,该订单的 CustomerID 会自动转换并传送到 CRMPortlet,同时自动提交查询请求;然后,请求获得的 Occupation 信息以及 OMPortlet 中的 Date 和 ProductID 会转换并传递到 BIPortlet,同样自动提交查询请求,从而仅仅通过一次鼠标点击动作,在 3 个 Portlet 中同时显示市场经理所需的全部信息.

1.2 Wrapper Portlet

上述场景中的基本前提是 OMS,CRM 及 BI 等应用系统已经集成到门户,即应用已经转换为 Portlet.在门户环境下,传统的集成技术仅仅考虑了功能集成,Web 服务和专有 API 的使用都属于此类.在这种方式中,尽管重用了业务逻辑,但是必须重新构造表示层,这导致若干潜在问题,如开发周期延长,成本增加等.为此,通用方法是采用类似于代理的 Wrapper Portlet 实现深度集成,使应用可以直接在门户环境中访问[3].Wrapper Portlet 采用 Bridge 设计模式[4],将应用作为一个黑盒处理,仅仅通过 URL 请求来访问应用,而应用则把 Wrapper Portlet 当作一个普通的 Web 客户.这意味着 Portlet 互操作应该在不对应用作任何改动的前提下实现.Wrapper Portlet 的实现关键是会话管理与 URL 地址改写.对 Wrapper Portlet 的讨论超出了本文的范畴,有关 Wrapper Portlet 的具体实现和关键技术请参考文献[3].

1.3 Portlet互操作需求

基于对上述场景及 Wrapper Portlet 实现技术的分析,可以得出 Portlet 互操作必须具有如下特征或能力:

(1) 基于前端的实现策略.Portlet 互操作的实现仅能通过扩展表示层的 HTML 标记片段以及增加外围支持模块完成.Portlet 所包装的已有应用不需要作任何修改.

(2) 互操作组件定义能力.可以通过指定 Portlet 标记片段上的元素定义互操作组件及其输入/输出数据,并支持多组候选输出.这种情况下,用户可以选择哪组输出用于实际的 Portlet 互操作过程.

(3) 基于语义的数据协作能力.数据是通过语义而非简单的类型匹配实现关联的,而且支持基于语义的

数据转换.互操作过程可以在给定条件满足时自动启动,或者由用户自己决定何时启动.参与到互操作过程的 Portlet 松散地耦合在一起,并可以灵活地重组.

2 Portlet 互操作模型

IEEE 对“互操作能力”的定义是:“两个或多个系统或组件交换信息并使用这些信息的能力”[5].INTEROP 将其扩展为更通用的形式:“系统或产品间无须用户特别处理而共同工作的能力”[6].作为表示层的中间件,门户提供了面向用户的集成能力.在门户环境中,对互操作的最通用需求是能够支持在 Portlet 之间交换数据.因此,Portlet 互操作的关键在于数据协作.文献[7,8]给出了基本协作模型.

定义 **1**. 协作模型定义为三元组(*E*,*L*,*M*),其中:*E* 代表参与协作的实体集合;*L* 为协作渠道;*M* 为协作模型所依赖的语义框架.

基于上述通用模型,我们给出了基于语义数据协作的 Portlet 互操作模型,如图 2 所示.其基本思想是:将参与互操作的 Portlet 抽象为称为 ShadowComponent 的组件,并作为基本的互操作实体.ShadowComponent 具有多个 Slot,它们对应到 Portlet 生成的标记片段的 HTML 元素,Slot 构成了互操作模型的交换数据.多个 ShadowComponent 的 Slot 通过映射到本体间接地联系起来.本体提供了语义层次的数据关联支持.我们基于 ECA 规则工作流技术[9]定义数据协作流程.当事件发生后,触发规则的执行并启动预设的动作,导致 ShadowComponent 向共享数据空间导出数据或者从共享数据空间导入数据,从而实现 Portlet 互操作过程.

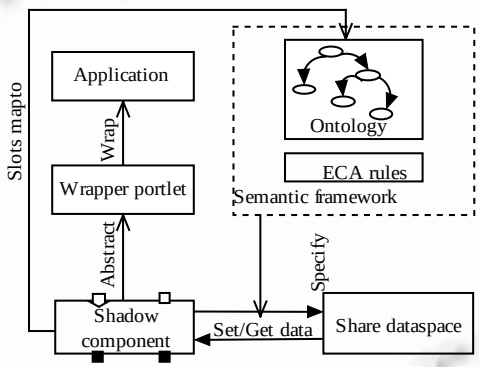

Fig.2 Portlet interoperation model

图 2 Portlet 互操作模型

定义 **2**. Portlet 互操作模型表现为门户环境中的语义数据协作模型,其定义为 *PI*=(*P*,*SC*,*SD*,*O*,*R*),其中:*P* 是参与互操作的 Portlet 集合;*SC* 是从 *P* 中 Portlet 提取的 ShadowComponent 集合,*SC* 与 *P* 为双射关系.*SC* 是语义协作的实体集合;*SD* 为 ShadowComponent 提供了共享数据空间,是互操作中数据协作的渠道;*O* 为 Portlet 互操作中的本体;*R* 是 ECA 规则集合,它定义了互操作过程中 ShadowComponent 执行有关操作的条件约束.*O* 与 *R* 共同构成了 Portlet 互操作模型的语义框架.

下面给出 ShadowComponent、操作原语、ECA 规则的明确定义.第 3 节将详细描述本体及语义数据关联.

2.1 ShadowComponent

定义 **3**. Slot 代表给定 Web 页面 *p* 中的一个 HTML 元素,它定义为一个三元组(*iepath*,*type*,*value*),其中:*iepath* 是定义在 *p* 上的信息提取路径;*type* 代表该 Slot 的类型,其值是本体中的概念;*value* 保存 Slot 当前的值.我们将在第 4.1 节对信息提取路径作详细讨论.

定义 **4**. ShadowComponent 是从 Portlet 中提取的组件,该组件仅包含一种方法,此方法可以基于给定输入数据给出对应输出数据.ShadowComponent 定义为五元组(*triggerSlot*,*IS*,*OS*,*inputProperty*,*outputProperty*),其中:*IS* 和 *OS* 均为 Slot 集合,它们代表 ShadowComponent 的输入与输出数据;*triggerSlot* 是一个特殊的 Slot,它定义了

ShadowComponent 的方法,其值为一个 URL.通常,该 URL 是一个“Submit”或“Click”动作,它通过提交当前的输入数据返回输出数据.*triggerSlot* 实际上体现了 ShadowComponent 从对应 Portlet 中提取的业务逻辑.*inputProperty*∈{*MANUAL*,*AUTO*,*TRIGGER*};*outputProperty*∈{*MANUAL*,*AUTO*}.它们决定了 ShadowComponent 的数据处理策略.表 1 给出了这些参数的描述.

Table 1 Descriptions of the input/output properties of a ShadowComponent

表 **1** ShadowComponent 输入/输出属性描述

Property	Description (Input)	Description (Output)
MANUAL	User deside when the data is transferred from share dataspace	User deside when the data is transferred to share dataspace
AUTO	Data is transferred from share dataspace as long as all input data needed is ready	Data is transferred to share dataspace automatically after trigger action
TRIGGER	Data is transferred as long as all input data needed is ready, then start trigger operation automatically	/

2.2 操作原语

Portlet 互操作模型中的操作原语包括两部分:Slot 操作原语和 ShadowComponent 操作原语.Slot 操作原语包括 *GetValue* 和 *SetValue*.ShadowComponent 操作原语包括 *Import*,*Export*,*GetExport* 和 *Trigger*.表 2 给出了这些原语的说明.

Table 2 Operation primitive descriptions

表 **2** 操作原语描述

Operation primitive	Belongs to	Description
GetValue	Slot	Get value from share data space
SetValue	Slot	Put current slot value to global share
Import	ShadowComponent	Invoke *GetValue* action of all *IS* slots of this ShadowComponent
Export	ShadowComponent	Invoke *SetValue* action of all *OS* slots of this ShadowComponent
GetExport	ShadowComponent	Aims at ShadowComponents without *triggerSlot*, invoke *GetValue* action of all *OS* slots
Trigger	ShadowComponent	Perform action defined in *triggerSlot*

2.3 ECA规则

Portlet 互操作的数据流转定义采用基于 ECA 规则的定义方法[9].

定义 4. ECA 规则是定义互操作逻辑和数据流转的基本要素,它决定了数据在 Portlet 之间的流转方式.ECA 规则定义为元组(*event*,*condition*,*action*),其中,*event* 的类型包括 *TriggerInput*,*TriggerOutput*,*InputDataReady* 和 *AskForInput*,每个事件都具有一个或多个参数,表示与该事件相关的对象或传递的数据.每个事件的描述见表 3.*condition* 是由 ShadowComponent 的 *inputProperty* 和 *outputProperty* 属性构成的逻辑表达式.*condition* 可以为空,此时意味着只要事件发生就执行相关事件;*action* 由 ShadowComponent 的操作原语构成.当 *action* 包括多个操作原语时,它们需要被顺序执行.例如 ECA 规则:

ON *InputDataReady*(*sc*1,*paraList*)

[IF *sc*1.*inputPorperty*==*TRIGGER*]

DO *sc*1.*Import*(*paraList*),*sc*1.*Trigger*,*sc*1.*Export*

Table 3 Event descriptions

表 **3** 事件描述

Event	Parameter table	Description
TriggerInput	(*sc*,*paraList*)	The *triggerSlot* of a wrapper Portlet corresponding to ShadowComponent *sc* is clicked, *paraList* contains the values of HTML elements corresponding to the *IS* slots of *sc*.
TriggerOutput	(*sc*,*paraList*)	Used by ShadowComponent without *triggerSlot*, represents the wrapper Portlet corresponding to *sc* submits its output data
InputDataReady	(*sc*,*paraList*)	Data for all input slots of a ShadowComponent *sc* is ready
AskForInput	(*sc*)	Wrapper Portlet corresponding to ShadowComponent *sc* requests to get input data from share dataspace

定义了当发生 *InputDataReady* 事件后,如果事件对应的 ShadowComponent 的 *inputProperty* 属性值为 *TRIGGER*,则首先导入数据,然后执行 *Trigger* 操作,并将所有输出参数导出到共享数据空间.

3 本体与 Portlet 互操作

3.1 互操作模型中的本体

本体是对共享概念模型的明确形式化规范说明[10].由于本体可以用来描述信息资源的语义,并实现这些资源的明确表达,因此,互操作是本体的重要应用之一.在互操作模型中,本体定义的概念模型描述了有关领域数据、元数据及与互操作过程相关的基本概念.本体在 Portlet 互操作中的作用有:① 类型匹配;② 语义转换.如图 3 所示,本体提供的这种基于语义的数据关联能力使松散耦合的 Portlet 互操作成为可能,提高了 Portlet 互操作过程的灵活性和重组能力.

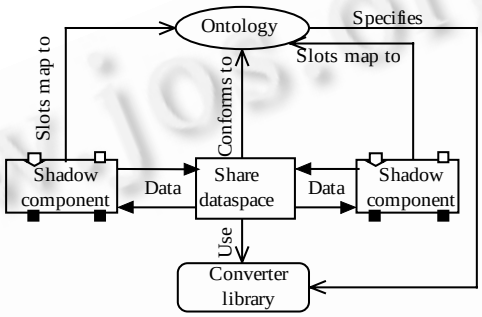

Fig.3 Ontology in Portlet interoperation model

图 3 Portlet 互操作模型中的本体

在类型匹配方面,来自异构应用的数据通过映射到本体中的概念而发生关联.通过这些关联关系,可以在参与互操作的组件间建立数据连接.ShadowComponent 使用 *Export* 操作原语向共享数据空间发布数据,而各 ShadowComponent 的 *IS* Slot 集合则从共享数据空间获取类型匹配的数据.由于应用的异构性,ShadowComponent 定义过程的独立性和数据类型的多样性,指向不同本体概念的两个数据也可能存在匹配关系.比如,两个分别指向 OrderID 和 PONo 的 Slot 可能都代表订单 ID.传统的类型匹配无法在这二者之间建立关联,本体则提供了优秀的解决概念异构的能力.

本体也使具有语义层次数据交换能力的 Portlet 互操作成为可能.数据之间的语义转换主要有两种类型:① 内容转换.考虑第 1.1 节中的示例场景,OMPortlet 与 CRMPortlet 中的 CustomerID 分别映射到本体中的 OMCustomerID 和 CRMCustomerID,它们的实例存在一一映射关系,为了使 OMPortlet 发送到共享数据空间的 CustomerID 能够为 CRMPortlet 所用,在共享数据空间将该 CustomerID 传递给 CRMPortlet 之前,必须先对其进行转换.CRMPortlet 与 BIPortlet 中的 Occupation 的转换也属于此类;② 分解转换.示例场景中,OMPortlet 中的订单日期为 Date 类型,而 BIPortlet 中年和月份分别为 Year 和 Month 类型,它们是 Date 类型的一部分.也就是说,一个 Date 类型的数据可以满足一个 Year 类型的数据订阅请求.我们将这种转换称为分解转换.

3.2 本体类型与数据语义关联

在具体实现中,Portlet 互操作模型中的本体定义参考了 W3C OWL-S 原子流程本体[11],并划分为 3 种类型:

(1) 基本本体.包括基本类型的定义,如数字、字符、URL、货币、日期、时间等;

(2) 领域本体.包括与各应用域相关的概念,如订单、客户、产品等;

(3) 基础设施本体.包括 Slot,ShadowComponent,Event,ECARule 和 Interoperation.它们是基于事件的 Portlet 互操作流程基础设施中的基本概念.该类本体主要由门户和 ECA 规则引擎内部使用.

我们定义了两个扩展的本体属性,以实现对数据语义关联的支持:*isComposedOf* 和 *canbeConvertedTo*.设

A,*B* 为本体中的概念,如果存在描述 *A isComposedOf B*,则表示 *A* 概念表示的数据可以满足 *B* 概念表示的数据.*A canbeConvertedTo B* 则表示 *A* 概念对应的数据可以转换为 *B* 概念类型的数据.每个 *isComposedOf* 和 *canbeConvertedTo* 都需要定义转换方法完成数据的变换.转换方法可以是简单函数、数据库访问或者外部的服务.领域无关的转换可以在基本本体中定义.与应用或者领域相关的转换可以定义并存储在外部的转换库中,并覆盖基本本体中定义的转换功能.

4 Portlet 互操作实现

我们在网驰平台[12]门户中间件 OncePortal v2.0 中对本文的方法进行了验证实现.OncePortal 是一个表示层中间件,可以方便地集成来自 Internet,Intranet 和应用系统的资源,并形成个性化页面.验证实现的关键在于 ShadowComponent 的构造和扩展 Wrapper Portlet,以提供符合 JSR168 规范[2]的 Portlet 互操作实现.

4.1 构造ShadowComponent

构造 ShadowComponent 的关键在于 Slot 的定义.本文的方法使用信息提取路径(information extraction path,简称 IEPath)标识与 Slot 对应的 HTML 节点.IEPath 对文献[13]提出的 HTML-Path 进行了删减和扩展,使其满足 Portlet 互操作的需求.它的定义采用了与 XPath[14]相同的基于位置定位的表达法.

IEPath 形式上表现为多个节点标识的连接序列,它定义了从文档根到指定元素的全路径.每个节点标识由标签名称和索引 *i* 构成,说明该节点是具有相同标签名称的第 *i* 个子节点. IEPath 的语法定义如下:

IEPath::=*Step*|*IEPath*/*Step*

Step::=*tagname*[*i*]|*text*(*regular-expression*)

IEPath 与 Xpath 的区别主要体现在以下两点:

(1) IEPath 允许具有变量的实例.这种表达用于处理 Web 页面具有多组输出数据候选的情况.例如,IEPath 表达式 *HTML*[1]/*BODY*[1]/*TABLE*[1]/*TR*[*i*]/*TD*[2]表示表格中每行的第 2 列节点的集合.从实现的角度看,这种情况需要在页面上构造链接来启动输出事件.

(2) 表达式 *text*(*regular-expression*)用于获取元素值的子串.设 *e* 是 IEPath *p* 对应的 HTML 元素,$r_{left}r_{capture}r_{right}$ 是 *e* 的值,则表达式 *p*/*text*(r_{left}(*)r_{right})表达的文本 *s* 为 $r_{capture}$.

IEPath 的定义以其所在的 HTML 页面为基础.由于 Portlet 通常包含多个交互页面,因此,Slot 的定义中需要考虑页面定位问题.在我们的实现中,认为 *IS* 中的 Slot 在 *triggerSlot* 所在的页面定义,而 *OS* 中的 Slot 则在 ShadowComponent 的 *trigger* 操作发生后,即 *triggerSlot* 对应的 URL 产生的返回页面中定义.*triggerSlot* 自身的定位以其值判断,这样,以 *triggerSlot* 为参照可以确定 *IS* 与 *OS*.ShadowComponent 可能没有 *IS* 和 *triggerSlot*,这表示 ShadowComponent 可以直接提交输出数据,此时,我们使用 *OS* 中 Slot 所在的页面的 URL 来标识该页面.

IS,*OS* 和 *triggerSlot* 定义完成后,将附加对应 Portlet 信息,形成 ShadowComponent 定义,保存在 ShadowComponent 注册表中,供互操作过程使用.对 Slot 的定义可以通过图形化界面用鼠标指定,也可以通过修改配置文件直接完成.

4.2 互操作过程

在门户中,用户与 Portlet 生成的标记片段交互,通过标记片段中包含的 URL 触发客户请求.按照 JSR168 规范[2],门户中有两种类型的 URL:ActionURL 和 Render URL.通常情况下,由 ActionURL 触发的客户请求转换为一个事件请求和多个显示请求.门户必须首先调用 *processAction* 方法处理事件请求,并等待该方法完成后,调用页面上所有 Portlet 的 *render* 方法执行显示请求.

为此,我们扩展 Wrapper Portlet 的 *processAciton* 方法,使 ECA 规则引擎和 ShadowComponent 实例能够参与到 Portlet 请求处理过程中,从而完成互操作过程.图 4 给出了 Portlet 互操作的协作图.我们实现了一个 ECA 规则引擎负责 ShadowComponent 实例和 ECA 规则实例的管理,并提供了共享数据空间.当 Portlet 容器加载 WrapperPortlet 时,会根据 ShadowComponent 注册表信息创建 ShadowComponent 实例及相关的 ECA 规则实例.

然后,这些实例传送给 ECA 规则引擎.Wrapper Portlet 生成的标记片段含有触发 Portlet 互操作的 ActionURL.当用户点击该链接后,请求转发给 WrapperPortlet,启动 *processAction* 方法.此时,对应事件被发送到 ECA 规则引擎,并触发 ShadowComponent 实例的动作.动作的执行可能会向共享数据空间写入数据,ECA 规则引擎基于本体和对共享数据空间的遍历判断是否发生新的事件,并启动指定动作.循环此过程,直到没有新的事件发生为止.此时, *processAction* 方法结束,同时将开始 *render* 方法.从实现的角度看,ShadowComponent 实例会保存标记片段信息,Wrapper Portlet 可以直接使用该信息生成返回给 Portlet 容器的标记片段.

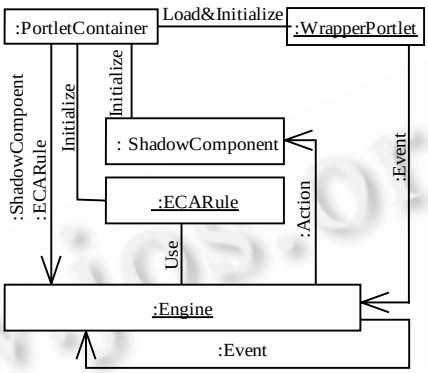

Fig. 4 Collaboration diagram of Portlet interoperation

图 4 Portlet 互操作过程协作图

对于第 1.1 节中的使用场景,互操作过程的事件和动作序列如图 5 所示.其中,*OmsSC*,*CrmSC*,*BiSC* 分别是与 *OMPortlet*,*CRMPortlet*,*BIPortlet* 对应的 ShadowComponent 实例.当用户在 *OMPortlet* 选中指定订单并提交请求后,*OMPortlet* 向 ECA 规则引擎发送 *TriggerOutput* 事件,引擎根据 ECA 规则启动 *OmsSC* 的 *GetExport* 和 *Export* 操作,其结果是用户选中的订单的 *CustomerID*,*ProductID* 和 *Date* 发送到共享数据空间中.引擎通过遍历共享数据空间,发现 *CrmSC* 的 *IS* 的全部 Slot 数据已就绪,向自己发送 *InputDataReady* 事件,并启动该 ShadowComponent 的系列动作.*CrmSC* 的输出会自动发送到共享数据空间,并进一步导致 *BiSC* 的动作.

A user clicked the icon in SMS portlet…
Event: *TriggerOutput*(*SmsSC*,*pt*1)
Action: *OmsSC.GetExport*
Action: *OmsSC.Export*
Event: *InputDataReady*(*CrmSC*,*pt*2)
Action: *CrmSC.Import*, *CrmSC.Trigger*, *CrmSC.Export*
Event: *InputDataReady*(*BiSC*,*pt*3)
Action: *BiSC.Import*, *BiSC.Trigger*, *BiSC.Export*

Fig.5 A sample event and action sequence of a Portlet interoperation

图 5 Portlet 互操作过程事件与动作示例

5 相关工作

目前已经存在多种方法用于解决 Portlet 之间的互操作问题.基于所采用策略的不同,这些方法可以划分为:基于共享的方法[2]、基于数据源的方法[15,16]和基于注释的方法[17].

基于共享的方法以 JSR168[2]为代表.JSR168 引入了“Portlet 应用”的概念来处理 Portlet 互操作问题.属于同一个 Portlet 应用的所有 Portlet 可以共享数据,从而实现这些 Portlet 间的互操作.但是在门户中,无论是通过包装已有应用生成的 Portlet,还是各独立开发的 Portlet,通常都属于不同的 Portlet 应用,从而使这种方法不能适用.本文方法则为所有的 Portlet 提供了全局的共享空间,无论其属于哪个 Portlet 应用,都可以完成数据交换.

Roy-Chowdhury 等人[15]和 Weinreich 等人[16]提出了基于数据源的 Portlet 互操作方法.该方法的基本思想是:通过定制的 JSP(Java server page)标签库或 XML 描述,使 Portlet 成为数据源.目标 Portlet 使用 WSDL(Web services description language)定义,通过定制的扩展来描述用于处理被传送数据的事件.在运行时,一个可以点击的图标插入到 Portlet 标记片段中.通过点击该图标,用户启动了从源 Portlet 到目的 Portlet 的数据流程.该方法的主要局限在于:目前对于标签库,编程接口及相关处理方法还没有被广泛接受的规范或标准,从而导致兼容性问题.此外,基于数据的方法和基于 API(application programming interface)的方法主要基于简单的类型匹配,而非语义匹配来实现数据交换,削弱了 Portlet 互操作的松散耦合特性.

Diaz 等人提出了另外一种基于注释的 Portlet 互操作方法[17],支持语义层次的数据交换.该方法中,Portlet 由各自的本体描述,然后,Portlet 扩展它们的 HTML 标记片段,加入该片段支持的流程信息,通过本体实例的映射获得 Portlet 互操作能力.但是,该方法要求 Portlet 开发者必须在开发阶段定义所有的元数据、结构及数据流转信息,而这些信息可能并不一定会被用到,因此会使 Portlet 开发复杂化,并增加了不必要的成本.

此外,绝大多数门户集成企业应用的场景中都是针对已经存在的应用的.由于维护、成本及技术等各种原因,必须能够在不对已有应用系统作修改的前提下,将其集成到门户.这说明,这些应用对应的 Portlet 之间的互操作也必须在不对原有系统作任何修改的前提下实现.但是,由于各系统并非针对互操作而设计开发的,因此,尽管上述 3 种方法都提供了 Portlet 间互操作的渠道,但是,Portlet 并不会去使用这个渠道,从而导致互操作仍然无法实现.

与上述方法不同,本文方法基于语义数据协作模型,使语义层次的 Portlet 互操作成为可能.首先,使用信息提取技术为 Portlet 构造 ShadowComponent,它的每个输入/输出映射到本体中定义的概念;之后,一组 ECA 规则被初始化,这些规则定义了数据在何时、以何种方式在 ShadowComponent 之间传递.本文的方法与上述 3 种方法的主要区别在于:① 本文的方法基于本体建立数据之间的语义关联,在此基础上使用 ECA 规则实现组件之间的语义数据协作.基于通用转换或领域相关操作的引入,使异构应用间复杂的互操作成为可能;② 本文的方法不需要参与到互操作过程的 Portlet 满足任何前提条件或者作出任何修改.而上述 3 种方法可能适用于在新开发的 Portlet 之间建立互操作过程,但是,对于已有应用集成的场景,这些应用不得不进行修改.

6 总结及进一步工作

门户提供了表示层的集成能力.在门户环境下,应用间的进一步集成问题实际上表现为 Portlet 互操作的问题.Portlet 互操作通过在已有相互隔离的应用之间建立通信渠道,可以方便地整合来自这些异构应用的数据,从而使门户环境下复合应用的构造成为可能.

本文描述了一种门户环境中基于语义数据协作的应用集成方法,它综合使用了数据协作、信息提取和本体技术.其基本思想是:将参与互操作的 Portlet 抽象为 ShadowComponent 组件;然后,通过将 ShadowComponent 的 Slot 映射到本体建立 ShadowComponent 之间的语义数据关联;最后,基于 ECA 规则实现基于语义数据协作的 Portlet 互操作过程.基于该方法和 Wrapper Portlet 技术,无须对已有应用作任何修改就可以实现门户中的企业应用集成.本文的主要贡献如下:

(1) 给出了一种基于语义数据协作的 Portlet 互操作模型;
(2) 分析了本体在 Portlet 互操作中作用及语义数据关联方法;
(3) 给出了基于 IEPath 的 ShadowComponent 定义方法;
(4) 给出了本文的方法符合 JSR168 规范的实现.

目前,门户环境中的应用集成还局限在数据流转层次,仍然存在若干没有解决的问题,如,涉及多个 Portlet 的事务处理、具有复杂控制逻辑的互操作等.这也是我们进一步的研究方向.

References:

[1] Clarke S. Standards for second-generation portals. IEEE Internet Computing, 2004,8(2):54−60.

[2] Java Community Process. JSR 168 portlet specification. 2003. http://www.jcp.org/en/jsr/detail?id=168

[3] Díaz O, Paz I. Turning web applications into portlets: Raising the issues. In: Cellary W, *et al*,. eds. Proc. of the 2005 Symp. on Applications and the Internet. Washington: IEEE Computer Society, 2005. 31−37.

[4] Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994.

[5] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York: IEEE, 1990. http://standards.ieee.org/reading/ieee/std_public/description/se/610.12-1990_desc.html

[6] Rosener V, Latour T, Dubois E. A model-based ontology of the software interoperability problem: Preliminary results. In: Grundspenkis J, *et al*., eds. CAiSE Workshops (3). Riga: Faculty of Computer Science and Information Technology, 2004. 241−252.

[7] Papadopoulos G, Arbab F. Coordination models and languages. In: Zelkowitz M, ed. Advances in Computers, Vol. 46. New York: Academic Press, 1998. 329−400.

[8] Malone TW, Crowston K. The interdisciplinary study of coordination. ACM Computing Surveys, 1994,26(1):87−119.

[9] Geppert A, Tombros D. Event-Based distributed workflow execution with EVE. In: Davies N, *et al*., eds. Proc. of the IFIP/ACM Int'l Conf. on Distributed Systems Platforms and Open Distribued Processing (Middleware). The Lake District: Springer-Verlag, 1998. 427−442.

[10] Gruber TR. Toward principles for the design of ontologies used for knowledge sharing. Technical Report, KSL-93-4, Stanford University, 1993.

[11] W3C Consortium. OWL-S: Semantic markup for Web services. 2004. http://www.w3c.org/Submission/OWL-S/

[12] TCSE ISCAS. Once platform. 2006. http://www.once.com.cn/

[13] Ito K, Tanaka Y. A visual environment for dynamic Web application composition. In: Ashman H, *et al*., eds. Proc. of the 14th ACM Conf. on Hypertext and Hypermedia. NewYork: ACM Press, 2003. 184−193.

[14] W3C Consortium. XML path language (XPath). 1999. http://www.w3.org/TR/xpath

[15] Roy-Chowdhury A, Ramaswamy S, Xu X. Using click-to-action to provide user-controlled integration of Portlets. 2002. http://www-128.ibm.com/developerworks/websphere/library/techarticles/0212_roy/roy.html

[16] Weinreich R, Ziebermayr T. Enhancing presentation level integration of remote application and services in Web portals. In: Prabhu G, *et al*. eds. Proc. of the IEEE Int'l Conf. on Services Computingb. Washington: IEEE Computer Society, 2005. 224−236.

[17] Díaz O, Iturrioz J, Irastorza A. Improving portlet interoperability through deep annotation. In: Ellis A, *et al*., eds. Proc. of the 14th Int'l Conf. on World Wide Web. NewYork: ACM Press, 2005. 372−381.

宋靖宇(1976－),男,黑龙江克东人,博士,助理研究员,主要研究领域为分布式计算,软件工程,门户中间件.

万淑超(1981－),女,博士生,主要研究领域为分布式计算,软件工程,门户中间件.

魏峻(1970－),男,博士,研究员,CCF 高级会员,主要研究领域为面向服务计算,中间件,移动计算,软件工程.