An HTML Fragments Based Approach for Portlet Interoperability

Jingyu Song, Jun Wei, and Shuchao Wan

Technology Center of Software Engineering
Institute of Software, Chinese Academy of Sciences
Beijing, 100080, P.R.China
{songjy, wj, wsc}@otcaix.iscas.ac.cn

Abstract. Presentation level integration now becomes an important and fast growing trend in enterprise computing and portals are the mainstream to realize it. However, there is not yet a definitive mechanism to achieve interoperability between the basic components of a portal i.e. portlets, whereby HTML data flows smoothly from one portlet to a neighboring one. This paper proposes an HTML fragments based approach to achieve portlet interoperability. Fragments are a block of HTML elements, which are generated by portlets and are used to aggregate a portal page. We first construct a presentation component, which is named as ShadowComponent, for each portlet involved in a portlet interoperation using its fragments, then define a data flow process between ShadowComponents using ECA rules, and finally drive such a process by creating events to fulfill data flow between ShadowComponents. As the fragments of a portlet are synchronized with their corresponding Shadow Component, such a process enables the portlet interoperation. Experimental results show that the proposed approach is effective in achieving portlet interoperability in portals.

Keywords: Portal, Porlet Interoperability.

1 Introduction

Presentation level integration now becomes an important and fast growing trend in enterprise computing [9] and portals are the mainstream to realize it. Portals enable the aggregation of interactive interfaces of different applications as components on the same web page [1]. Portlet is the basic component of a portal, which represents an interactive web mini application and is deployed on a portal server [7].

A portal typically decorates the HTML fragment returned by a portlet with a title and several buttons, such as minimize, maximize and edit etc., then aggregates all fragments together into a portal page. Though such unconstrained aggregation is useful since applications are simultaneously rendered in the same page and users see comprehensive information in a more convenient way, further integration capability is surely desired. Information contained in a portlet may be required as the input in other portlets. The information has to be manually copied from source to target portlets. Such manual interactions may lead to frustration, low productivity, and inevitable

J. Indulska and K. Raymond (Eds.): DAIS 2007, LNCS 4531, pp. 195–209, 2007.

mistakes. Therefore, an effective mechanism for portlet interoperation is needed. Unfortunately, currently available standards such as JSR168[7] and WSRP[11] support no further integration of portlets than being displayed on the same page.

This paper proposes an HTML fragments based approach to achieve portlet interoperation in portals. Rather than resorting to back-end solutions, we support a pure front-end approach. A presentation component, which is named as ShadowComponent, is constructed for each portlet involved in an interoperation using the fragments produced by the portlet. Then an interoperation process, which uses ShadowComponents as its nodes, is defined using event-condition-action (ECA) rules. An ECA rule defines when and how the input/output data of a ShadowComponent are received from or sent to a shared data space. Because the fragments are synchronized with their corresponding ShadowComponents, such a process achieves the interoperation between portlets. As the approach is based on the fragments generated by portlets only, there is no need of modifications for portlets to take part in an interoperation.

The rest of this paper is organized as follows: Section 2 presents related work. Section 3 defines the requirements concerning portlet interoperation in portals based on a typical scenario first, and then analyzes the inefficiencies and drawbacks of the approaches that implement interoperation at different layers of a portlet based on a general portlet architecture and points out that using fragments to achieve portlet interoperation is a more reasonable solution. Our approach is proposed and discussed in detail in section 4, 5 and 6. A practical example is also discussed in section 6. Finally conclusions and future work are given in section 7.

2 Related Works

A variety of mechanisms for portlet interoperation have been proposed, which can be classified as application-based, datasource-based and annotation-based.

The application-based approach, which is proposed by JSR168[7], introduces the notion of “portlet application” that allows distinct portlets to share a common piece of information to achieve portlet interoperation. However, a portal normally frames portlets from distinct portlet applications, which prevents the data from being exchanged.

Both approaches presented by Roy-Chowdhury et al.[14] and Weinreich et al.[16] can be classified as datasource-based since the authors propose the use of a custom JSP tag library or XML descriptions to enable a portlet to be a data source. The target portlet is defined in a WSDL file with a custom extension to describe the actions, which can consume data transferred from other portlets. However, the description-based approach may cause compatibility problem, as there is no agreement yet on how to standardize this mechanism.

Diaz et al. propose an annotation-based portlet interoperation approach that supports semantic data transfer[2]. In that approach, portlets are characterized by their ontology. Then portlet fragments extend their markups with information about the supported process. Portlet interoperability is achieved through the mapping of the ontology concepts. However, this approach relies on the cooperation of the markup producer who has to embed the underlying information structure into the fragments in

the development phase. Moreover, the approach further requires that the operation defined in the specification should be extended.

Furthermore, in many scenarios, a portal is used to integrate existing web-based applications. An application may be integrated into a portal without modifications because of maintenance, cost, or technical reasons. Therefore, a portlet interoperation should also be achieved without modifications to the corresponding applications. In such situations, though all above three approaches provide some kinds of mechanisms to transfer data between portlets, the portlet may not use them because the portlets or back-end applications were not designed and developed to be used in an interoperation context, which makes interoperation hard to be achieved.

3 Problem Statement and Analysis

3.1 A Scenario

We use the following scenario to analyze portlet interoperation requirements. Consider a marketing department of a motor corporation. Three are three applications developed and deployed: Order Management System(OM), Customer Relationship Management System(CRM) and Business Intelligence System(BI). Each application has been wrapped into a portlet, OMPortlet, CRMPortlet and BIPortlet respectively using the method proposed in [3].

To analyze the market situation of cars and to find out the potential customers, the marketing manager built a Market Analysis portal page containing the above three portlets. The marketing manager has to interact individually with each portlet on the page and key in data manually. For example, to get the customer details of an order, the manager must copy the CustomerID of specified order from the OMPortlet to the CRMPortlet's entry textbox, and submit the query by clicking on the "Submit" button. If the manager needs further to do a data mining to find out the sale status of such a car model in the community with the same occupation as the customer in this month, he/she has to copy the ProductID, Date from OMPortlet and Occupation from CRMPortlet to BIPortlet's corresponding entry textbox again. As shown in Fig.1, the whole process is very fussy and error prone, which affects the fluency of analysis process greatly.

According to the IEEE Standard Computer Dictionary, interoperability means "the ability of two or more systems or components to exchange information and to use the information that has been exchanged"[6]. The essential function of portlet interoperation is to provide a mechanism that would facilitate portlet

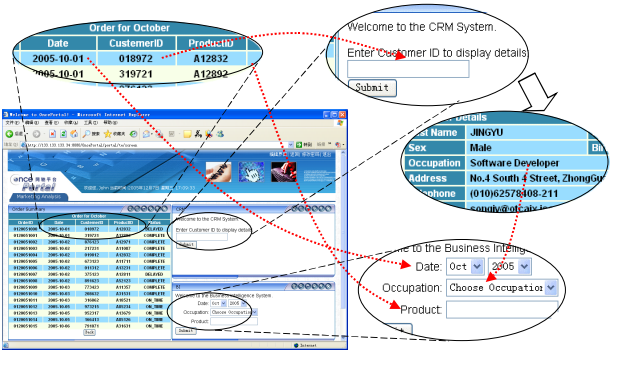

Fig. 1. A scenario of portlet interoperation

interactions by enabling easy transfer of compatible data between portlets. Given the above example, a better data flow is shown as follows: by one click in OMPortlet, the CustomerID is transferred to CRMPortlet; then CRMPortlet submits a query request automatically with the received CustomerID; and then the Occupation in the response page and Date, ProductID in OMPortlet are transferred to BIPortlet automatically; again a request is submitted. With such an automated mechanism, all required information could be displayed in the three portlets simultaneously only by one mouse click.

Thus, we can define the basic requirements of portlet interoperation as follows.

1. A portlet need not to be modified to take part in an interoperation. That requirement enables interoperation between portlets within one portlet application, portlets of different portlet applications and even remote portlets.
2. Supporting multiple outputs and 1:n communication. A portlet may have a set of output candidates. In such a case, a user can choose which output data is used. Data from one portlet may be simultaneously sent to a number of destination portlets.
3. Supporting portlet wiring. An interoperation process can be started automatically or manually. Portlets involved in an interoperation are loosely coupled and can be decomposed and re-composed easily.

To make it a general and platform independent approach, one additional requirement is defined as follows:

4. Support standards based implementation. The use of standards allows reuse of standard compliant portlets and enables the independency from a particular portal.

3.2 Achieving Interoperability at Different Layers of a Portlet

Usually, portlets employ a similar layered architecture as general web applications, as shown in Fig.2. The architecture consists of four layers: resource layer, service layer, orchestration layer and presentation layer.

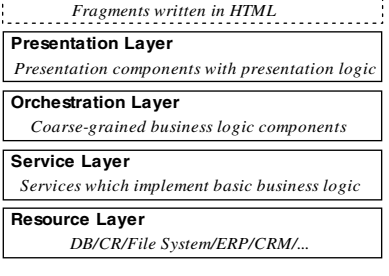

Fig. 2. Layered Portlet Architecture

Resource layer contains the resources that a portlet uses, such as database, content repository, and file system, etc. Service layer consists of basic services that are developed on top of resource layer, which represent business logic software units that satisfy the enterprise business requirements. Orchestration layer assembles services to coarse-grained business components. Presentation layer creates the graphical view of the portlet, and interacts with portal users. It is important to point out that presentation layer is not the user interface presented by markup language such as HTML. Presentation layer is a part of a portlet. It has its own model and process logic.

According to the analysis of section 3.1, the problem we concerned with is how to achieve the association and transfer of HTML elements, which are located on fragments, between portlets. It should be noted that we could achieve such a goal by working on all these four layers. That is because the four layers of a portlet are related

with each other. When the model or data of a lower layer change, the data or model of the layer above it will also change. However, the approach implemented on each layer has some deficiencies or drawbacks that are list as follows:

1. Achieving portlet interoperability at resource, service and orchestration layer are indirect solutions to the problem. To use these solutions, the portlet designers have to consider interoperation requirements, such as which HTML elements in a fragment are involved in the interoperation, besides the requirements of each layer at design time, which increases the problem complexity.
2. Whatever layers we used to implement portlet interoperation, we have to know the technical details of the portlet. For example, to implement interoperation at resource layer, we have to know the data schema details of the resource the portlet used. That also increases the complexity of portlet interoperation. Moreover, not all information of each layer of a portlet is accessible in enterprise environment, e.g. a portlet may be produced by wrapping an existing web-based application.
3. There are currently no acceptable and standard methods to invoke or to share the components of the orchestration and presentation layer of a portlet, which makes it difficult to achieve portlet interoperability at these two layers directly.

Thus, we have to find out another approach beyond such layers. Noted that all portlets use HTML to describe their fragments and our goal is also to achieve the association and transfer between HTML elements, we hope to find out a method based on such HTML fragments that are produced by each portlet. Such an approach at least has the following two merits:

1. It is a general and platform-independent solution. Because only HTML fragments are employed, the approach can be used in different scenarios, no matter which applications the portlet belongs to, how the portlet is designed and developed. That makes possible that the approach can be implemented on different portal servers.
2. There is no need of the knowledge of the technical details of the portlets involved in the interoperation. The approach does not care about the technical details such as service interfaces, how to invoke a component, etc. That is also to say, there is no need to modify a portlet to make it involved in an interoperation.

There are mainly two key problems in such an approach: how to describe the user interfaces of a portlet i.e. the fragments produced by the portlet; how to define associations and how to transfer data between HTML elements. We will propose our approach to portlet interoperation based on the answer of these two questions.

4 Reference Model for Portlet Presentation Layer

Moreno et al. proposed a reference model for portlet[10]. In such a model, the presentation layer consists of six main sub models: Conceptual, Navigation, Presentation, User, Context and Adaptation, as shown in Fig.3a. The Conceptual model encapsulates the information handled by the rest of the models at the presentation layer. The Navigation model describes the application navigational requirements building the navigational structure of the portlet. The Presentation model captures the presentational requirements in a set of HTML elements. The User

model describes and manages the user characteristics. The Context model deals with device, network, location and time aspects. The Adaptation model is used to obtain appropriate web content characteristics and target markup.

For modeling the presentation layer of a portlet, we need at least its Conceptual, Presentation, and Navigation models. However, as we do not know the exact internal details of a portlet, we can only reconstruct the presentation components using fragments by a reverse engineering way. So we propose a simplified presentation model in our approach, which describes the most important characteristics of the presentation layer of a portlet, as shown in Fig.3b. The simplified presentation model consists of three sub models: **Element**, **Location** and **Interaction**. Element is a simplified Conceptual model, which describes what types of elements are located on the fragments. Location is a corresponding model to Presentation, which defines the locations for each elements described in Element. Interaction is a simplified Navigation model, which defines the interactive relationships between elements, e.g. a customer's name can be obtained by submitting a *CustomerID*.

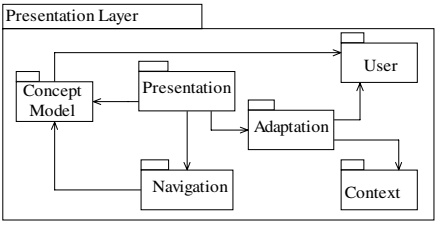

Fig. 3. (a) Presentation model of a portlet

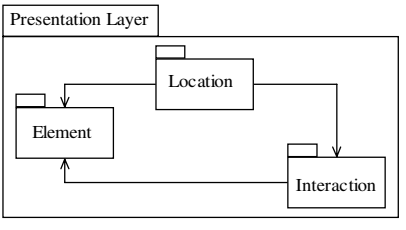

Fig. 3. (b)Simplified presentation model

5 Portlet Interoperation Model

Papadopoulos et al.[13] and Malone et al.[8] gave the basic model of coordination.

Definition 1. A coordination model can be viewed as a triple (E, L, M), where **E** represents the entities being coordinated, **L** the media used to coordinate the entities, and **M** the semantic framework the model adheres to.

In this paper, we propose a portlet interoperation model based on the above generic coordination model, as shown in Fig.4.

Definition 2. A portlet interoperation model is the coordination model in a portal context, it is defined as a tuple (PF, SC, SD, O, R), where **PF** is the set of fragments of the portlets that participate in an interoperation. **SC** is the set of ShadowComponents corresponding to PF. **SD** provides a shared data space for portlet interoperation. **O** represents the ontology used in the interoperation. **R** represents the ECA rule set that defines the conditions about when and how to execute a data flow. From a coordination model point of view, SC is the entity of the semantic coordination model, O and R together form the semantic framework of the portlet interoperation model, and SD is the data coordination media.

A ShadowComponent is constructed for each portlet, which takes part in an interoperation, using its fragments. A ShadowComponent usually has several slots that represent the HTML elements located in portlet fragments. The ShadowComponent keeps synchronized with its corresponding portlet fragments during the whole interoperation process. Each slot has its type that maps to a concept of the ontology, which achieves semantic data type match between slots. Finally, ECA rules define a data flow process, which uses ShadowComponent as its nodes. An ECA rule specifies when and how a ShadowComponent receives matched data or sends data to shared data space. Because the portlet fragments are synchronized with the corresponding ShadowComponent, the execution of such a data flow process transfers an HTML element value on a portlet fragment to a neighboring one, thereby achieving portlet interoperability.

In the following subsection, the detailed definitions of Shadow Component, Operation Primitives and ECA rule are presented. Then, we will further explain the proposed interoperation model by discussing the implementation of such a model in a real portal server.

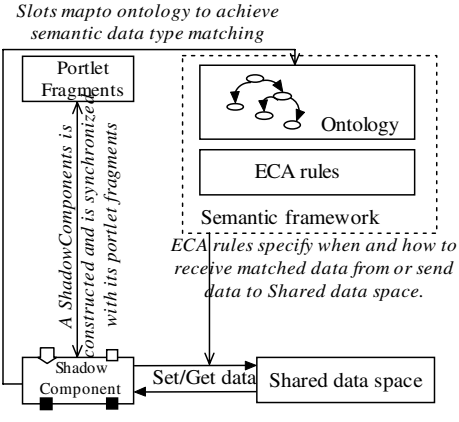

Fig. 4. Portlet interoperation model

5.1 ShadowComponent

Definition 3. A slot represents an HTML element in a given portlet fragment FP, it is a triple (path, type, value) where **path** is the information extraction path, which we proposed in [15], of the element in FP. An information extraction path is a concatenation of node identifiers along a path from the root to the specified element, thereby specifying the location of an element. **type** represents the slot type with its value constrained to the concept set defined in the ontology, **value** stores the current value of the slot.

Definition 4. A ShadowComponent is a component constructed using portlet fragments and is synchronized with the fragments of the portlet. A ShadowComponent is defined as a tuple (triggerSlot, IS, OS, inputProperty, outputProperty, status) where Both **IS** and **OS** are slot set, representing input and output data of the ShadowComponent. **triggerSlot** is a special slot whose value is a URL, which indicates the interaction relationship between IS and OS. Usually the URL represents a "submit" or "click" action that returns output data using current input data. **InputProperty**∈{MANUAL,AUTO,TRIGGER} and **outputProperty**∈{MANUAL, AUTO}, which decide the data process policy of the ShadowComponent. The descriptions of these values are showed in table 1. **status** is a BOOL variable, which is used to indicate if all input data needed could be obtained from a shared data space.

Table 1. A summary of input/output properties

Property	Summary(Input)	Summary(Output)
MANUAL	A user decides when the data are loaded from Shared data space	A user decides when the data are sent to Shared data space
AUTO	Data are loaded from Shared data space as long as all input data needed is ready	Data are sent to Shared data space automatically if they are available in fragments
TRIGGER	Data are loaded if all input data needed is ready, then a request is submitted automatically after the fragment is displayed in the client side browser	/

ShadowComponent is the realization of the simplified reference model of a portlet presentation layer, which is proposed in section 4. The types of input and output data form the Element model; the paths of input data and output data form the Location model; whereas the triggerSlot, IS and OS together form the Interaction model.

5.2 Operation Primitives

The operation primitives in portlet interoperation model consist of two parts: slot operation primitives and ShadowComponent operation primitives. Slot operation primitives include GetValue and SetValue. ShadowComponent operation primitives include Import, Export and SetStatus. Table 2 gives the detail.

Table 2. Descriptions of Operation Primitives

Operation Primitive	Belongs to	Description
GetValue	Slot	Load matched data from Shared data space
SetValue	Slot	Send current slot value to Shared data space
Import	ShadowComponent	Invoke GetValue action of all IS slots of the ShadowComponent
Export	ShadowComponent	Invoke SetValue action of all OS slots of the ShadowComponent
SetStatus	ShadowComponent	Set the status of the ShadowComponent

Table 3. Descriptions of Events

Event	Para Table	Description
SlotDataReady	(slot)	There is a match data for the given slot in Shared data space
TriggerOutput	(ShadowComponent)	A user starts a request to output data manually
InputDataReady	(ShadowComponent)	Data for all input slots of a ShadowComponent *sc* is ready
AskForInput	(ShadowComponent)	A user starts a request to input data from Shared data space

5.3 ECA Rules

We employ an event-based architecture[5] to define data flow process between ShadowComponents.

Definition 5. ECA rule is the fundamental metaphors for defining and enforcing data flowing logic, it is a tuple (event, condition, action) where the possible values of **event** include SlotDataReady, TriggerOutput, InputDataReady and AskForInput. Each event has parameters indicating to whom the event is oriented. Details of each event are shown in table 3. **condition** is a logic expression that is composed of inputProperty and outputProperty of a ShadowComponent. **action** is composed of ShadowComponent operation primitives. condition could be *null*, which indicates the action should be executed as long as the event occurs. When an action consists of several operations, the operations should be executed serially. For example, the ECA-rule

ON InputDataReady(sc1) [IF sc1.inputPorperty = = TRIGGER]
DO sc1.Import, sc1.SetStatus(TRUE)

indicates that when an event InputDataReady happens, if the inputProperty of the corresponding ShadowComponent is TRIGGER, the ShadowComponent will first import data, then set status to TRUE.

6 Implementation

We have validated our approach by extending OncePortal portal system of ONCE platform[12]. OncePortal is a JSR168 and WSRP compatible portal, which can integrate different resources and aggregate them into personalized page. Since our implementation is based on the Portlet and WSRP specifications, it can be easily migrated to any JSR168 compatible portal server.

6.1 Constructing ShadowComponents

The key to construct a ShadowComponent is slot definition. The information extraction path used to define a slot is specified in the context of a fragment. Because there are usually several fragments returned by a portlet during the whole interoperation process, we have to consider in which fragment the slot is defined. We use the following two methods in our implementation:

- In default, we assume that the slots of IS and triggerSlot of a ShadowComponent are defined in the first fragment produced by a portlet. If a ShadowComponent has no input slots, then slots of OS are defined in the first fragment. In most practice scenarios, these assumptions can be satisfied, whereas they decrease the implementation complexity greatly.
- Adding fragment marks. If the above assumption cannot be satisfied, then we need to do some modification to the portlet, which adds marks to the fragment to indicate that it has IS or OS slots. Such marks can be simply added as the properties of an HTML element on the fragment or provided as HTML annotations.

A ShadowComponent can be constructed visually by specifying some portions on the portlet fragments to work as IS/OS slots through mouse operations or can be pre-configured using configuration file.

6.2 InteroperationFilter

InteroperationFilter is one of the most important components in our approach. Fig.5 gives the location of InteroperationFilter during the whole interoperation process.

When a user submits a request in a browser, it is received by portal servlet. We define two types of portal request in a portlet interoperation process: normal request and interoperation request..

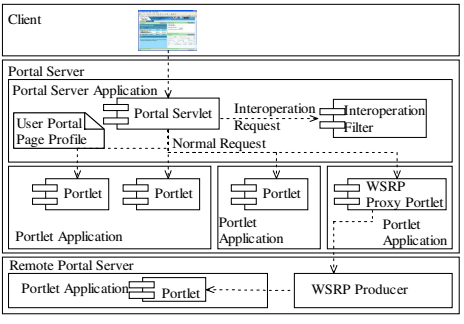

Fig. 5. Portlet interoperation process

For a normal request, portal servlet uses a pre-defined user page profile to find which portlets are needed to build the requested page. It then forwards the request to the corresponding portlets. Each portlet returns a fragment, which is aggregated with a general page frame and the fragments returned from the other portlets to form the final portal page. In common portals, the page will be returned to the browser and waiting for next request at this time. However, to achieve portlet interoperability, we first transfer the fragments returned by each portlet to InteroperationFilter, which rewrites each fragment based on the interoperation related information. Then portal servlet uses such modified fragments to assemble the final page and returns it to the browser. Based on the fragment and the input/output properties of the corresponding ShadowComponents, there are two types of process:

1. **There are output parameters on the fragment, the value of outputProperty is MANUAL.** In such a case, InteroperationFilter modifies the fragment so that to insert icons before each output parameter. By clicking on an icon, a user can output a parameter or the whole of the parameters that the portlet provides. From technical point of view, such a click submits an interoperation request, which embeds the output parameters as its request parameter.
2. **There are output parameters on the fragment, the value of outputProperty is AUTO.** InteroperationFilter informs the ShadowComponent to export its output parameters to Shared data space. The value of outputProperty is not allowed to be AUTO, if the ShadowComponent has an output parameter whose path has variable, preventing the situation that which parameters to be used cannot be decided.

After finishing the process, new data are added or updated to Shared data space, which may create new events, such as InputDataReady, etc. Such events then trigger certain actions, which may start the three types of process for input parameters:

1. **There are input parameters on the fragment and the value of inputPorperty is MANUAL.** In such a case, if all input parameters of the ShadowComponent can be

obtained from Shared data space, then inserts an icon into the fragment, which will submit an interoperation request when it is clicked on.

2. **There are input parameters on the fragment and the value of inputProperty is AUTO.** In such a case, InteroperationFilter retrieves data from Shared data space and fills the input slots of the ShadowComponent automatically. Different with the process when inputProperty value is MANUAL that an icon will be inserted only when all input parameters are ready, InteroperationFilter will try to fill each slot as long as a matched data can be obtained from Shared data space for it.
3. **There are input parameters on the fragment and the value of inputProperty is TRIGGER.** In such a case, the process is similar to the case when inputProperty is MANUAL i.e. it is only be processed when all input parameters needed are ready in Shared data space. After the input parameters are filled into the fragment, a block of JavaScript is further added to the element that is specified by triggerSlot of the ShadowComponent, whose function is to submit the page automatically after the page is displayed in the browser.

On the other hand, an interoperation request is processed by InteroperationFilter directly. InteroperationFilter creates events according to the request, which ultimately results in the data flowing between ShadowComponents and Shared data space based on ECA rules. There are two types of interoperation request:

1. **A user outputs data manually** i.e. a user clicks the icon that is inserted by InteroperationFilter for the fragment whose corresponding ShadowComponent's outputProperty is MANUAL during the process of normal request. In such a case, InteroperationFilter creates event TriggerOutput that results in the execution of Export operation of the ShadowComponent, which exports data to Shared data space. Also, other events may be created because the adding or updating of data.
2. **A user requires to fill data manually** i.e. a user clicks the icon that is inserted by InteroperationFilter for the fragment whose corresponding ShadowComponent's inputProperty is MANUAL. In such a case, InteroperationFilter creates event AskForInput, which results in the execution of Import operation of the ShadowComponent, thereby loading data from Shared data space.

The definition information for ShadowComponents is stored in portal page profiles for each user, while not the portlet related profiles, so that to ensure that given the same portlet, a user can decide whether that portlet takes part in an interoperation process and how the interoperation happens.

Whatever type of a request that is received, InteroperationFilter initializes ShadowComponents or synchronizes the ShadowComponents with corresponding fragments, i.e. to update the input/output parameters using the received fragment, based on current interoperation definition information. After all event and action are processed, another synchronization from ShadowComponents to fragments is processed i.e. to update the fragments using current input/output data of the corresponding ShadowComponents. Moreover, fragments are cached to ensure the interoperation request can be processed by InteroperationFilter only.

6.3 Interoperation Process

InteroperationFilter is the only component that interacts with portal servlet. However, the whole interoperation process is supported by several components together.

Besides InteroperationFilter, other important components include ECA rule engine, Shared data space, ShadowComponent instances, etc. The collaboration diagram is shown in Fig.6.

InteroperationFilter receives fragments and user portal page profiles from portal servlet and then initializes ShadowComponents and ECA rules based on such information. New events are created by InteroperationFilter and Shared data space. When receiving these events, ECA rule engine sends actions to certain ShadowComponents based on current ECA rules.

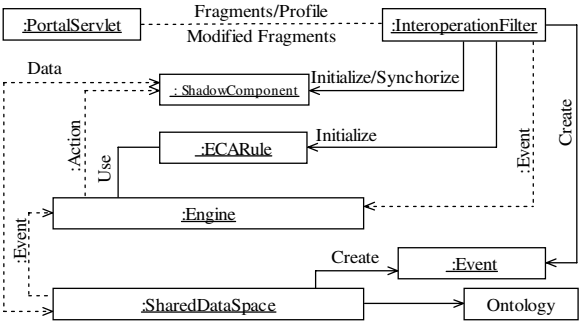

Fig. 6. Collaboration Diagram of Portlet Interoperation

The events created by InteroperationFilter are mainly related to user interactions such as TriggerOutput, AskForInput, whereas the events created by Shared data space are mainly data-related such as InputDataReady, SlotDataReady. The execution of an action may create new events that result in new actions. When there is no event created, InteroperationFilter does the synchronization from ShadowComponents to fragments and decides if scripts should be added to fragments based on their properties such as if status is TRUE. All modified fragments and other cached fragments that are not involved in the interoperation then are returned to portal servlet to aggregate the final portal page. InteroperationFilter will wait for next request.

For the scenario described in 3.1, the events and actions sequence of an interoperation process is depicted in Fig.7. OmSC, CrmSC and BiSC are corresponding ShadowComponents for OMPortlet, CRMPortlet and BIPortlet. Comprehensive information are displayed in a portal page only by one mouse click.

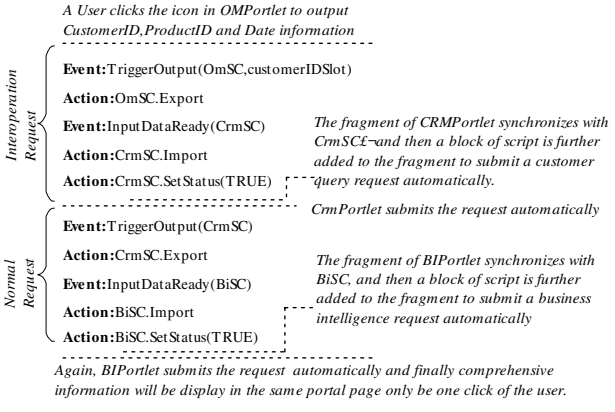

Fig. 7. Sequence of events and actions of an interoperation

6.4 A Practical Example

Our framework opens a new vista to the integration of applications and services in portal context, which makes possible portal-based composite applications.

Fig.8 shows a composite application that is constructed in OncePortal using our proposed portlet interoperation approach. The composite application is composed of three portlets: TripSchedule, WeatherForecast and FlightSearch. The TripSchedule portlet is an internal information system that shows the user's trip schedule in the near future. The Weather Forecast portlet provides weather information for a given city and the FlightSearch portlet provides flight information from the user's current city to a destination city. They are constructed by wrapping two Internet web sites: eLong Flight[4] and Yahoo Weather [17] using the approach proposed in [3].

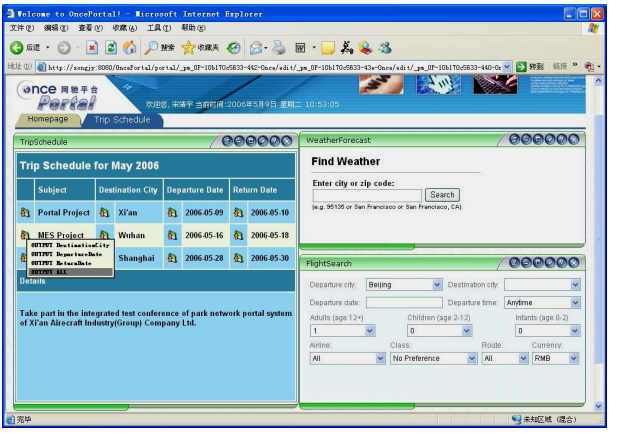

Fig. 8. (a)Trigger a portlet interoperation

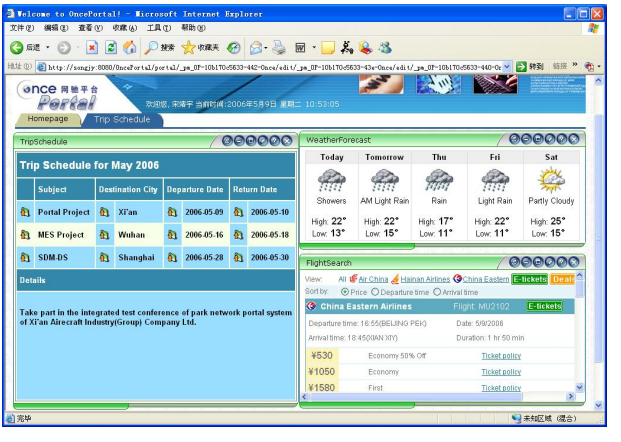

Fig. 8. (b) After the portlet interoperation

We configure the ShadowComponents for the three portlets manually by defining the configuration file. The corresponding ShadowComponent of TripSchedule has two output parameters: DestinationCity and DepartureDate. WeatherForecast has one input parameter: City. FlightSearch has two input parameters: DepartureDate and DestinationCity. When a user clicks on the icon before each row of the trip schedule table, which is generated automatically by InteroperationFilter, and chooses OUTPUT All (Fig.8a), WeatherForecast and FlightSearch portlets will receive DepartureDate and DestinationCity from TripSchedule, and show the weather and flight information for the specified city and date (Fig.8b).

7 Conclusion

Portals provide presentation level integration capability. Portlet interoperability makes possible portal-based composite applications, which enable users to easily fuse data

and processes from multiple existing stove-piped systems into a unified solution at presentation level.

This paper describes an HTML fragments based approach for portlet interoperability. We first construct a presentation component, which is named as ShadowComponent, for each portlet involved in a portlet interoperation using its fragments, then define a data flow process between ShadowComponents using ECA rules, and finally drive such a process by creating events to fulfill data flow between ShadowComponents. As the fragments of a portlet are synchronized with their corresponding ShadowComponents, such a process enables the portlet interoperation. The proposed approach fulfills all functional and non-functional requirements defined in Section 3.1. The most important features of our approach are: (1) it is a general and platform-independent solution; (2) no knowledge of the internal workings of the interoperating portlets is required. That is also to say, a portlet need not to be modified to take part in an interoperation process.

Acknowledgments

This paper was supported by the National Natural Science Foundation of China under Grant No.60673112; the National High-Tech R&D Plan of China under Grant Nos.2006AA01Z19B, 2006AA01Z161; the National Key Technology R&D Program of China under Grant No. 2006BAH02A08.

References

[1] Clarke, S.: Standards for Second-Generation Portals. IEEE Internet Computing 8(2), 54–60 (2004)

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

[3] Díaz, O., Paz, I.: Turning Web Applications into Portlets: Raising the Issues. In: Proc. of the 2005 Symposium on Applications and the Internet, pp. 31–37. IEEE Computer Society, Washington, DC (2005)

[4] eLong Flight (2006) http://www.elong.net/flights

[5] 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, pp. 427–442. Springer, Heidelberg (1998)

[6] Institute of Electrical and Electronics Engineers: IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries, New York (1990)

[7] Java Community Process. JSR 168 Portlet Specification (2003) http://www.jcp.org/en/jsr/detail?id=168

[8] Malone, T.W., Crowston, K.: The Interdisciplinary Study of Coordination. ACM Computing Surveys 26(1), 87–119 (1994)

[9] McDonough, B.: Enterprise Portal Survey. An Examination of Business Processes Driving Adoption (2004) http://www.marketresearch.com/map/prod/1045547.html.(2004)

[10] Moreno, N., Romero, J.R., Vallecillo, A.: Incorporating Cooperative Portlets in Web Application Development. Workshop on Model-driven Web Engineering (MDWE 2005) (2005)
[11] OASIS. Web Services For Remote Portlets Specification (2003) http://www.oasis-open.org
[12] Once Platform (2005) http://www.once.com.cn
[13] Papadopoulos, G., Arbab, F.: Coordination Models and Languages. In: Zelkowitz, M. (ed.) Advances in Computers, vol. 46, pp. 329–400. Academic Press, New York (1998)
[14] Roy-Chowdhury, A., Ramaswamy, S., Xu, X.: Using Click-to-Action to Provide User-Controlled Integration of Portlets (2002) http://www7b.software.ibm.com/wsdd/library/teacharticles/0212_roy/roy.html
[15] Song, J., Wei, J., Wan, S., Huang, T.: Extending Interactive Web Services for Improving Presentation Level Integration in Web Portals. Journal of Computer Science and Technology 21(4), 620–629 (2006)
[16] Weinreich, R., Ziebermayr, T.: Enhancing Presentation Level Integration of Remote Application and Services in Web Portals. In: Proc. IEEE Int'l Conf. on Services Computing(SCC05), pp. 224–236 (2005)
[17] Yahoo Weather (2006) http://weather.yahoo.com