OASIS WSIA Technical Committee

Requirements Document
Use Case Report: Coordinated Producers

Version <1.2>

Revision History

Date	Version	Description	Author
05/Mar/2002	1.0	Coordinated Producers	Dan Gisolfi, Graeme Riddell, Alan Kropp, Eilon Reshef, Gil Tayar, Rex Brooks, Ravi Konuru, Keven Brinkley, Aditi Karandikar, Monica Martin, Rich Thompson, Charlie Wiecha
11/Mar/2002	1.1	Added discussion of basic activity flow through coordinated updates of multiple presentation services	Charlie Wiecha, Monica Martin
14/Mar/2002	1.2	Factored discussion into general flow and sub-flows with example approaches to coordination.	Shankar Ramaswamy, John Lucassen, Charlie Wiecha

Table of Contents

Use Case Report: Coordinated Producers

1. Definition of the Coordinated Producers use case

Definition: Use cases in which the Consumer uses information returned either from Producers it controls, or from interaction with the End-user, to change the state (setting properties, calling operations) of multiple Producers in order to create a unified End-user experience among them. The Consumer establishes connections between the Producers it is coordinating. Possible means include executing procedural code in the Consumer that explicitly coordinates a known set of Producers or interpreting a declarative binding specification that controls which properties on each Producer are linked with which Properties on other Producers.

1.1 Brief Description

The goal of coordinating multiple Producers from the Consumer is to create a unified End-user experience from a set of independently-authored Producers.

The business value of such an integrated user experience is to reduce the effort required by the End-user to accomplish tasks with the system by reducing the need to manually drive multiple Producers to the appropriate and inter-related states required by a unified task.

Producers may need to be coordinated in their (1) presentation states (selected page, tab, field), and (2) in their data states, i.e. to avoid the need for the user to enter redundant data across multiple Producers, and to convert data from the format required by one Producer to that of another Producer.

Examples of coordinated Producers from the WSIA scenarios include:

[Traveler's checks] As the user selects amounts of Travelers Checks, a field in the left in a separate WSIA service saying "Purchases So Far" is being updated.

[Traveler's checks] Providing a different navigation bar for Travelers Checks, and connecting its presentation and data state to the rest of the Travelers Checks application.

[Financial charting] The Consumer adds an additional Producer for custom interaction with the End-user in selecting iChart preferences. As the End-user interacts with this Producer, changes in its state need to be reflected in changes to the preferences/properties of the iChart Producer. The Consumer is responsible for ensuring that these changes are propagated to the other Producers, including iChart. Possible means include: application-specific code (ie. procedural implemented coordination) or by indicating through a binding specification the connections between the properties of the various Producers so that the changes can be propagated by some kind of Consumer middleware, or by other means.

[Multimedia sports portal] Syndication of synchronized (coordinated in WSIA terms) content from different sources. Different media needs to be synchronized to provide (1) a high level of interactivity and (2) seamless End-user experience. To what degree are streaming media types in scope for WSIA? The coordination requirement is independent of the media type so this is a valid use case nonetheless, but which media types are in scope should be considered explicitly.

2. Actors

There are three actors in this use case:

- Producer: one or more WSIA web services
- Consumer: an application that acts as a “container” that instantiates and controls interaction with the Producers on behalf of End-Users.
- End-User: a person who interacts directly with the output of the Consumer

3. Flow of Events

3.1 Basic Flow

Consumer composes page out of individual producer fragments (note that a subset of the producers may be used on any particular consumer page). In the process of doing so, the consumer will rewrite references to all user actions within the markup to ensure it receives all of them.

WSIA requirements:

- Standards for enabling composition of producer markup (i.e. producers need to produce legal fragments that can be composed).
- Standards for flagging user action references in producer markup.
- Standards that allow augmenting action references with originating producer identity for disambiguation at the consumer.

User interacts with page and selects a particular action.

Consumer receives the user action and does one of the following:

- Delegates the action to the originating producer (which includes the consumer in case the action resulted from a page fragment produced locally at the consumer).
- Performs some pre-processing and post-processing before and/or after delegating to the originating producer, which may include setting properties on the originating producer
- Performs some pre-processing and post-processing before and/or after invoking one or more actions and/or settings properties at any of its producers including the originating producer. This may include observing property changes at any of its producers and setting properties on that producer or any other producer as a result
- Processes the action itself without delegating to the originating producer or any of the other producers.

WSIA Requirements:

- Standards for describing producer action semantics including the inputs they process.
- Standards for describing producer property semantics including the values associated with them
- Standards for describing the circumstances under which the consumer can invoke a specific producer action. [NOTE: this requirement fits better in the “Orchestrated” use-case]
- Standards for describing the circumstances under which the consumer can set a specific producer property and the circumstances under which the consumer can expect changes in a specific producer property. [NOTE: this requirement fits better in the “Orchestrated” use-case]
- Standards for describing how the consumer can maintain a consistent state with a producer if it decides to bypass a specific producer action. [NOTE: this requirement belongs in the “Orchestrated” use-case]

Consumer obtains new markup fragments from producers (note that this set may be different from the ones on the previous page) to compose the page (similar to Step 1). Cycle continues.

3.2 Alternative Flows

3.2.1 Generalized XFORMS update processing model for propagating property changes

The basic flow of activity in this approach to the Coordinated use case is shown in Figure 4.1. This case corresponds to the example where an action bar provided by default in the Stock Plot Producer is to be replaced by the Consumer with one more suited to the specific way in which the Stock Plot Producer is to be used in this context.

For example, given knowledge of the End-users stock portfolio, perhaps the Consumer will allow only certain stock symbols to be charted. Perhaps a limited range of dates will be allowed. Perhaps a given stock symbol is to be added to the chart for comparison purposes if alternative investment decisions are being considered. For all of these reasons, a customized action bar is required offering either a subset of functions of the default action bar, or a customized set of functions tailored by the Consumer given it's knowledge of the particular End-user context.

In advance of the End-user interacting with the coordinated set of Producers, the Consumer must establish the interrelationships among them that will coordinate their presentation and data states. The basic mechanism for doing so is to define relationships among the properties of each Producer in such a way that changes to one Producer's properties will cause corresponding changes to one or more other Producer's properties. Following the distribution of these property change events, the Consumer will request the updated output from all changed Producers, and will reconstruct either the complete or incrementally updated page to return to the user.

Various alternative mechanisms for propagating property change updates and reconstructing the page (entirely or incrementally) are considered in the alternative subflows.

In the basic flow, we assume that the Consumer hasa binding specification that controls which properties on each Producer are linked with which Properties on other Producers. These binding specifications may resemble XLINK-based expressions. In general, there may be transformations on each link which can be used to change the format of properties and convert them from the defined schemas for the properties of the source Producer into the defined schemas for the properties of the target Producer.

The remaining steps of the basic flow are identified numerically in Figure 4.1, and are:

1. The Consumer requests the current output from each of its Producers and constructs the composite page to be returned to the End-User.
    - This process follows the sequence described in the Embedded and Customized user cases in that actions and URLs need to be rewritten by the Consumer as output flows through it toward the End-User to allow subsequent user interaction to be delegated to the appropriate Producer.
    - The manner in which each Producer's output is composed with that from other Producers and from the Consumer is not addressed by this use case (or indeed, even by WSIA). Alternative layout policies and technologies may be employed as desired by the designer of the Consumer.

- WSIA may need to specify the constraints on how output may be emitted by Producers in order that it can be combined into valid composite pages. In the absence of a general web-based framework for compound documents, (see W3C interest group on container-component standards) most fragment composition frameworks adopt a set of markup conventions that allow intermediaries to assemble complex pages from the various fragments emitted by each Producer. WSIA should seek to reuse these conventions where possible and avoid introducing a new set of its own. In particular, if WSRP will define a standard set of markup conventions, WSIA should consider reusing those conventions.

2. The End-user's interaction is returned to the Consumer.
   - End-user interactions in WSIA are defined as updates or operations on the output of a component, i.e. require validation by the service before resulting in updates to its properties.
   - These semantics allow the individual WSIA service to determine the manner in which its property values are to be computed (for example to apply data validation), and to introduce private or intermediate values to support a particular interaction with the End-user.
   - In Figure 4.1, the End-User interaction is returned via an HTTP POST message. A converter will map this interaction into a Soap message invoking an operation on the Consumer. The Consumer then uses the URL/action encoding discussed in the Embedded and Customized user cases to determine to which Producer the interaction should be delegated.
3. The Consumer then forwards the user interaction to that Producer by calling the indicated operation (typically invokeAction).
4. During the processing of the user interaction, the Producer may update any of its properties. Note that the manner in which user interaction messages are related to properties is not defined by WSIA. Individual Producers may employ various programming models which help to define this linkage. Some examples may include:
   - XFORMS. The Producer's output may be controlled by an XFORMS view, i.e. specified using the XFORMS UI widget set. The elements in the XFORMS view are bound to its model using XPATH expressions. The XFORMS model then would appear as all or part of that Producer's property set.
   - Servlet/JSP: The Producer's output would be generated by the execution of a set of JSP tags embedded in a page. Some or all of the Producer's properties could correspond to the values in the data object passed to the JSP to control the generation of its output. Returning updated property values would be set by the servlet.
   - Struts: This programming model follows closely on the servlet/JSP model above. The Struts form bean may give a more direct way of mapping the WSIA properties onto the particular programming model in this case.

5. Changed property values in the Producer originally receiving the End-user interaction trigger updates to other Producers whose properties are in turn dependant on the changed values. Several alternative subflows for propagating changed values to other dependent Producers are discuss below.
6. In Figure 4.1, applying the property changes in step (6) on the stock plot proxy triggers the remote invocation of the stock plot service. This step is not, of course, visible to the Consumer, and would not be required in the case that the stock plot service is running locally at the Consumer.
7. In general, changes to one set of a Producer's properties may have side effects and make changes in other properties.
8. The action bar is finally updated with side effects from its original property changes being applied to other dependent properties.
9. Output can now be generated by the Consumer from all changed Producers.
    - Using the Producer dependency graph from above, the Consumer requests output from all Producers having been notified of property changes during the update cycle.
    - If a producer is able to determine that no additional links into it exists for further property change notifications, it may generate and return its output immediately. Two alternatives exist for propagating this output back to the Consumer: (1) along the return path from property change notifications, and (2) directly to the Consumer using an additional (output)change notification connection. [If the review of whether or not a Producer can ever safely make this determination determines it can’t, this bullet should be deleted.]
10. The consumer returns the resulting page to be returned to the End-user, applying all URL/action rewriting steps as in the Embedded and Customized user cases.

This approach for distributing change notifications to other Producers who depend on changed property values is a generalization of the update processing model used in XFORMS to update its model when elements are changed.

The property change processing model builds a graph of all Producers depending directly on indirectly on the changed Producer. The graph is ordered in terms of the degree of dependency, i.e. the number of intermediate Producers who are in turn dependent on the changed Producer (in-degree).

The processing model does not depend on knowing the details of how each Producer will map changed inputs to changes in other properties. Each Producer is opaque to the processing model in this sense.

A limited form of loops among Producers is supported. In this way, the action bar in Figure 4.1 may trigger changes on the stock plot properties, and then eventually be notified of corresponding changes to other of the action bar properties that result. For example, by setting a particular stock plot type in step (4) the action bar service may be notified subsequently in step (9) that only certain date ranges are now valid for a related property. This related property will then be used to constrain the values for the user to select in a corresponding pull-down of the action bar.

Updates to WSIA properties are applied in batch, i.e. all properties for a given target Producer are updated at one time in order to minimize the number of network roundtrips required to propagate change notifications to all directly or indirectly dependent Producers.

The backward dependencies from the stock plot Producer to the action bar Producer are modeled with separate binding expressions so that they can be included in the overall graph of update dependencies and evaluated in the appropriate sequence. In Figure 4.1, side effect changes stock plot properties flow back to the action bar service through any required transformations on the binding expressions.

Infinite looping in property change notifications is prevented by computing the network of Producer dependencies only at the start of the update processing model. [when is this? [It theory the dependency graph can be computed when the binding statements are parsed. Logically it is produced (likely by subsetting the larger one) when a change notification is being processed exactly as per XFORMS] When the application is compiled, linked, or loaded? Whenever the dependency graph is changed?] In this case, all of the steps outlined above would be visible and executed once. The final updating of the action bar Producer with side effects from its change to the stock plot Producer would not trigger another iteration of the update processing cycle since it represents the final step of the original cycle, leaving the list of Producer dependencies empty. [Not positive this statement is correct. It seems to imply that the Consumer doing the coordination has to know all the impacts of sending the change notification to the Stock Plot Producer and that is not a likely scenario.][**How do we resolve this?]

3.2.2 *Procedural coordination for propagating property changes*

Alternative approaches to synchronizing changed property values among multiple Producers include doing so through application specific code implemented at the Consumer. In this approach, no additional infrastructure is required to declare dependencies among Producers, nor to determine the order in which Producers will be informed of changes to their values that result from changed properties on other Producers. This logic is expressed directly by the application designer responsible for the Consumer.

3.2.3 *Flow composition for propagating property changes*

The sequence of steps discussed above, as application level code at the Consumer, for propagating changed property values among Producers can clearly also be described as a control and/or a data flow among Producers. Thus web services tools for expressing and composing flows are an additional means of providing the application-specific specifications required at the Consumer for interdependencies among Producers.

3.2.4 *Stream-oriented approach to Producer coordination*

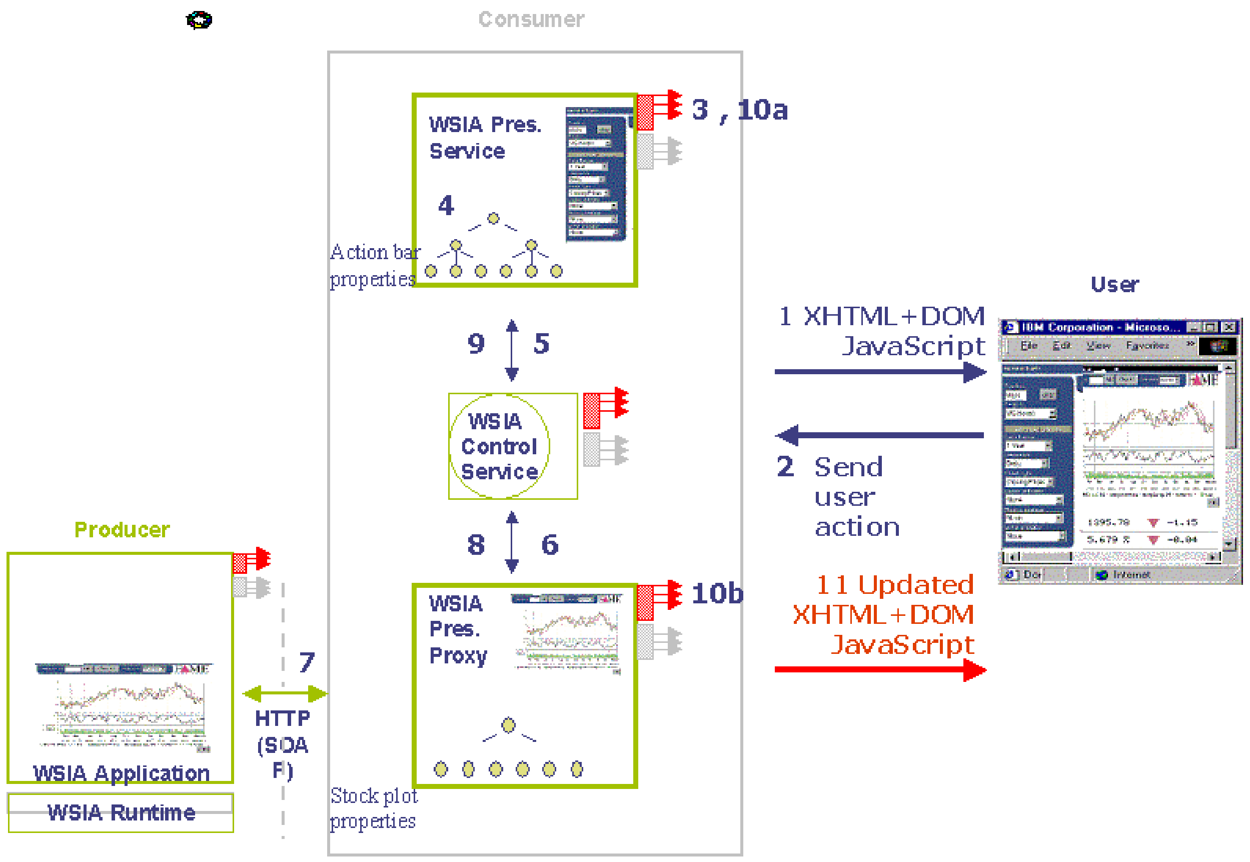
As in the Customized use case, there are in general both property/operation oriented means for Producer coordination, and stream-oriented approaches to coordination. In the property approach discussed above, the Consumer uses changes in Producer property values, independent of the markup, to determine corresponding changes in properties on other Producers -- in advance of requesting the updated output from all changed Producers.

In the stream-oriented approach, the Consumer can intercept messages originating either from the Producer or from the End-user, extract appropriate data from either the output or the input messages, and use that data to (1) to set properties or invoke operations on the original or another Producer, or (2) to implement the required function locally at the Consumer.

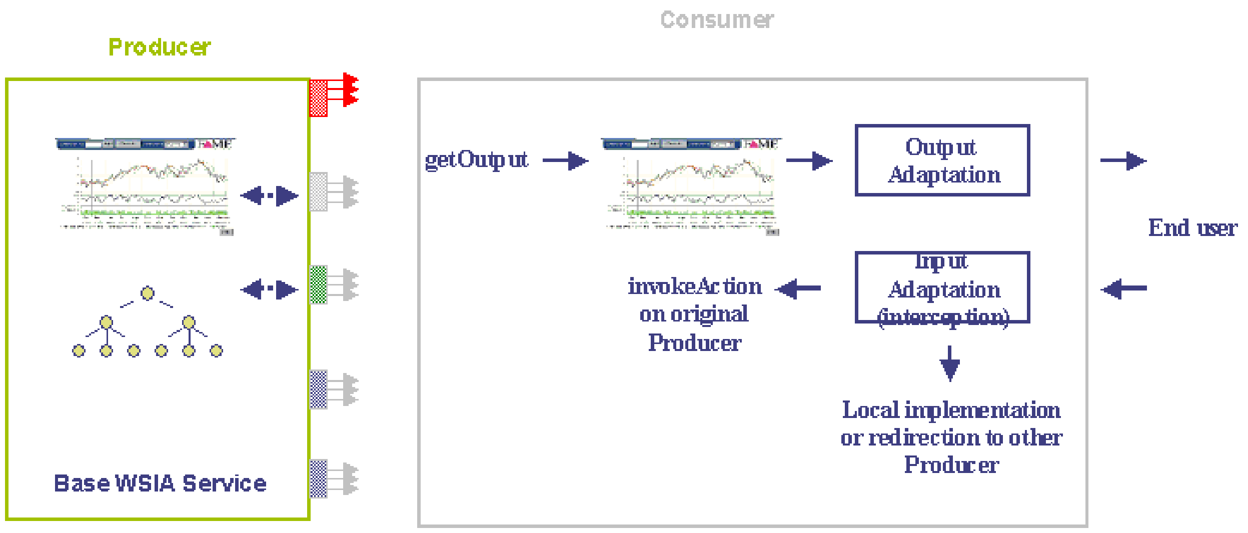
Figure 4.2 illustrates this approach to service coordination.

4. Diagrams

4.1 Relationship between Producers and Consumers in the Coordinated Producers Use Case



4.2 Stream-oriented approach to coordination



Consumer
Producer
Base WSIA Service
getOutput
Output Adaptation
Input Adaptation (interception)
invokeAction on original Producer
Local implementation or redirection to other Producer
End user

5. PreConditions

[A precondition (of a use case) is a textual description of any constraints or dependencies that must be satisfied prior to entry of the use case.]

5.1 < Precondition One >

6. PostConditions

[A postcondition (of a use case) is a textual description of any constraints or dependencies that must be satisfied after termination of the use case.]

6.1 < Postcondition One >