CHAPTER **8**

Protocols in an Open Hypermedia Framework

Navigation Template: Specifying The Control Flow

Intent

Provide a high level specification for the control flow of the hypermedia system to make clear what parts must be extended to achieve a certain effect. This specification is independent of particular information repositories, viewer applications or navigation resolution algorithms.

Analysis

An object-oriented framework is a set of co-operating classes that makes up a reusable design for a given problem domain. This is in some sense opposed to a software library, which is a set of routines where the code may be reused in different implementations. It is widely recognised that object-oriented frameworks aim at design reuse, while software libraries aim at code reuse.

One of the important differences between software libraries and frameworks is the so-called *Hollywood principle* ("don't call us, we'll call you") [Cotter,Potel'95]. This principle is the basis to understand how an object-oriented framework promotes design reuse and has to do with the definition of the control flow. With a software library approach, a software engineer writes code that calls the routines supplied by the library, so it is the software engineer that is defining the control flow. With the framework approach, a software engineer writes code that is installed in the framework and the framework calls that code.

The Hollywood principle states that a framework manages the control flow, which implies that the definition of the control flow is an essential ingredient of a framework's design specification. So, an important question to answer is how much of the control flow is incorporated in the design.

So far ([interoperability], [navigation], [resolver], [editor], [loader], [events], [meta-objects], [meta-meta-objects]), we have established design specifications for domain level, system level and configuration level tailorability. The emphasis of these design specifications lay on the definition of the elements constituting the framework structure and the role those elements play in that structure (i.e. the "Contract" section). As far as control flow concerns, this is too detailed to be useful. As a result, the control flow of the overall system is unclear, which makes it hard to understand what part of the system must be extended to achieve a certain effect.

Note that —according to the Hollywood principle— most of the functionality of the system is defined in the classes supplied by the software engineer tailoring the framework for a particular application. To attain a clear specification of the overall control flow, we must ensure that the specification is independent of particular information repositories, viewer applications or navigation resolution algorithms.

Problem

According to the Hollywood principle, provide a specification for the control flow of the hypermedia framework.

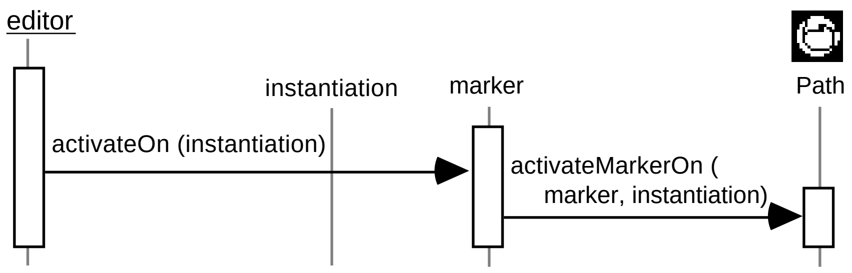
Solution

Define an infinite loop, with a body that defines how control is transferred from one object to another. Incorporate important messages only and leave the detailed protocols are for the contracts. Base the infinite loop on the navigation operation, which is the heart of any hypermedia system. Call it *navigation template*, to emphasise that is a description of a navigation operation omitting details of the participants involved.

Contract

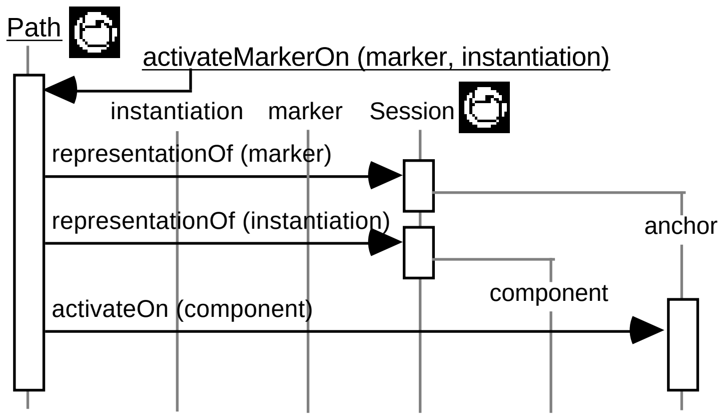
(See [object interaction diagrams] for a short survey of the main elements in an object interaction diagram)

Step 1: Selection of origin point



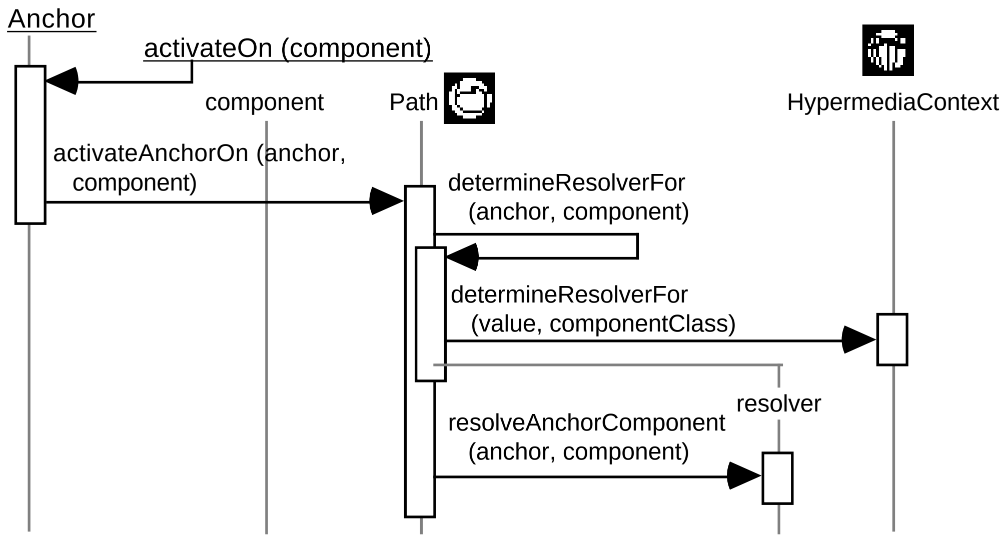
All navigation starts with a selection of the navigation source by the viewer application. This event is detected in the editor object and initiated by sending the `activateOn` message to a marker with an associated instantiation object as parameter. The marker transfers control to the Path meta-object by means of the `activateMarkerOn` message.

Step 2: Identification of origin point


Path
activateMarkerOn (marker, instantiation)
instantiation
marker
Session
representationOf (marker)
representationOf (instantiation)
anchor
component
activateOn (component)

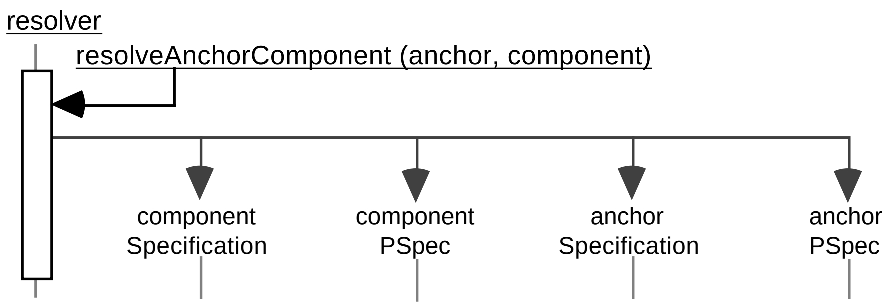
On receiving the `activateMarkerOn` message, the Path infers the anchor and component associated with the given marker and instantiation by sending the `representationOf` message to the Session meta-object. Afterwards, the Path transfers control to the anchor object by means of the `activateOn` message (with the component as parameter). Note that this scheme allows to start navigation without presentation layer objects, which is important for automatic analysis of static hypermedia structures.

Step 3: Start resolution process


Anchor
activateOn (component)
component
Path
HypermediaContext
activateAnchorOn (anchor, component)
determineResolverFor (anchor, component)
determineResolverFor (value, componentClass)
resolver
resolveAnchorComponent (anchor, component)

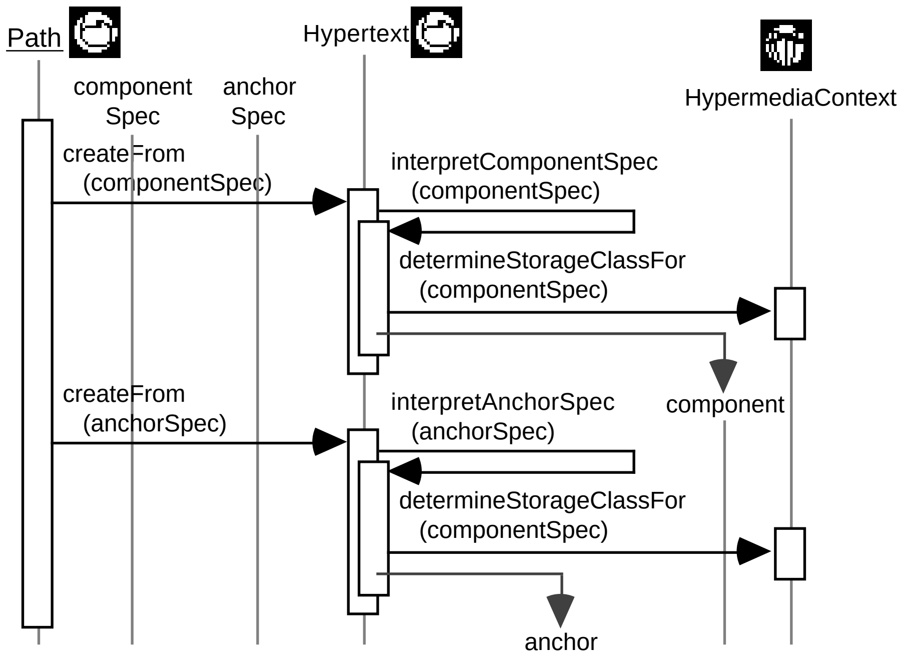
On receiving the `activateOn` message, the anchor transfers control back to the Path meta-object by means of the `activateAnchorOn` message. The Path then identifies the resolver that is supposed to handle the resolution request by sending the self message `determineResolverFor`. As part of this identification process, the Path requests the HypermediaContext meta-meta-object for the name of an installed and active resolver. Afterwards, the Path requests the identified resolver for the navigation targets by means of the `resolveAnchorComponent` message.

Step 4: Resolution process

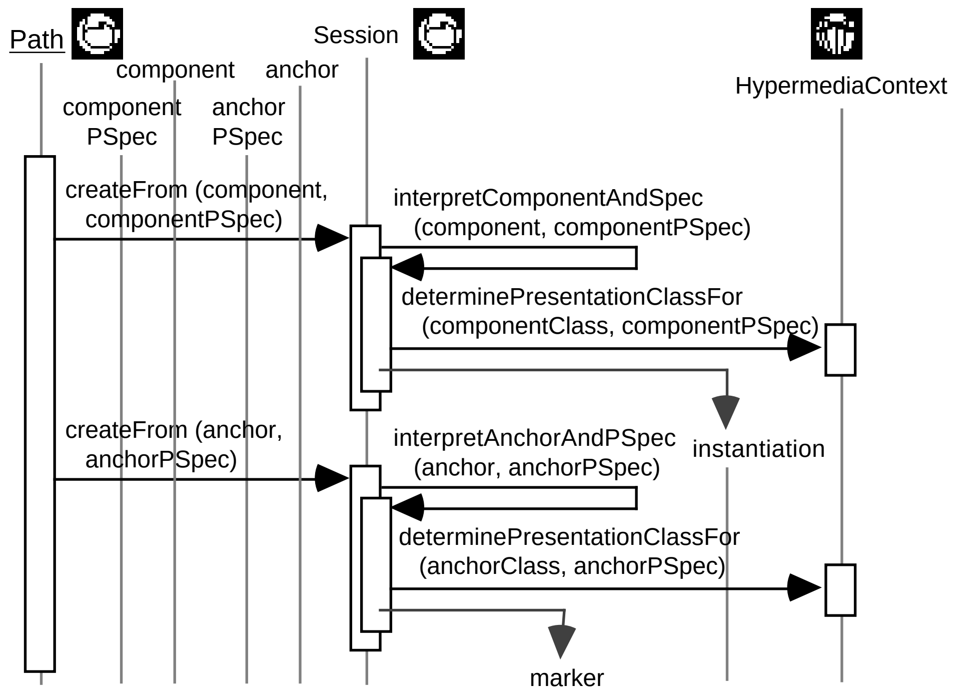


On receiving the `resolveAnchorComponent` request, the resolver returns a collection of quadruples to the Path object. Each quadruple in the collection is a specification for one navigation target.

Step 5: Target identification

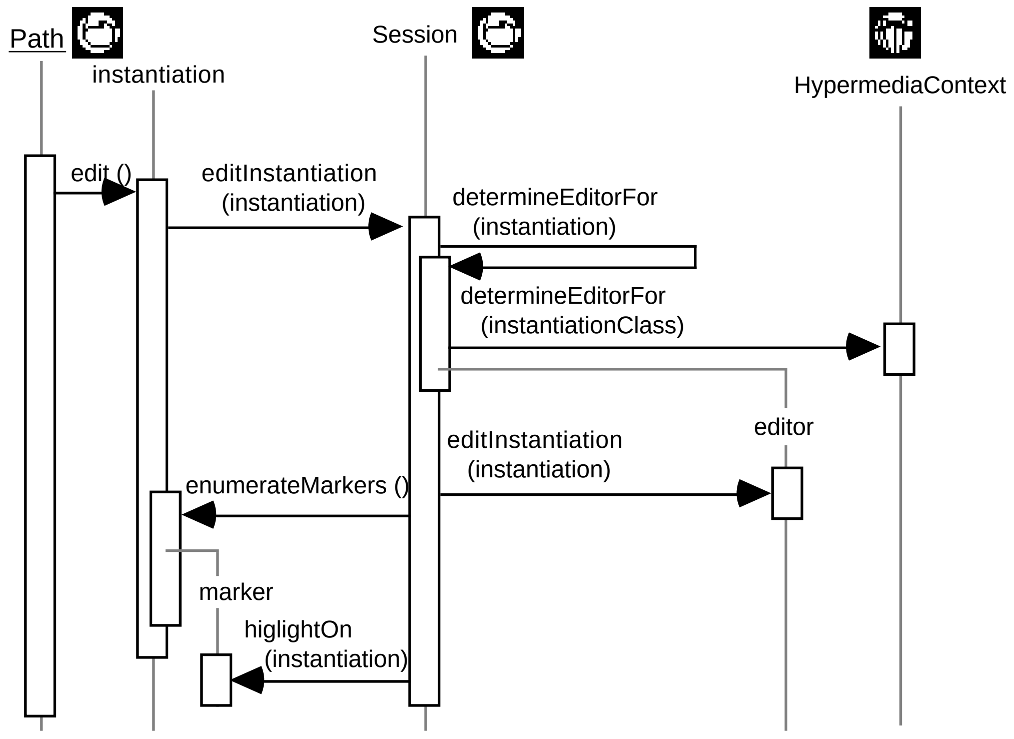

Path
Hypertext
HypermediaContext
component Spec
anchor Spec
createFrom (componentSpec)
interpretComponentSpec (componentSpec)
determineStorageClassFor (componentSpec)
component
createFrom (anchorSpec)
interpretAnchorSpec (anchorSpec)
determineStorageClassFor (componentSpec)
anchor

On receiving the quadruples specifying the target locations, the Path turns them into objects representing those locations. For each quadruple, the Path object asks the Hypertext meta-object to create a component (anchor) by sending the `createFrom` message. The Hypertext object interprets that specifier with a self send of `interpretComponentSpec` (`interpretAnchorSpec`), which in turn requests the HypermediaContext object for the class to instantiate by means of the `determineStorageClassFor`.

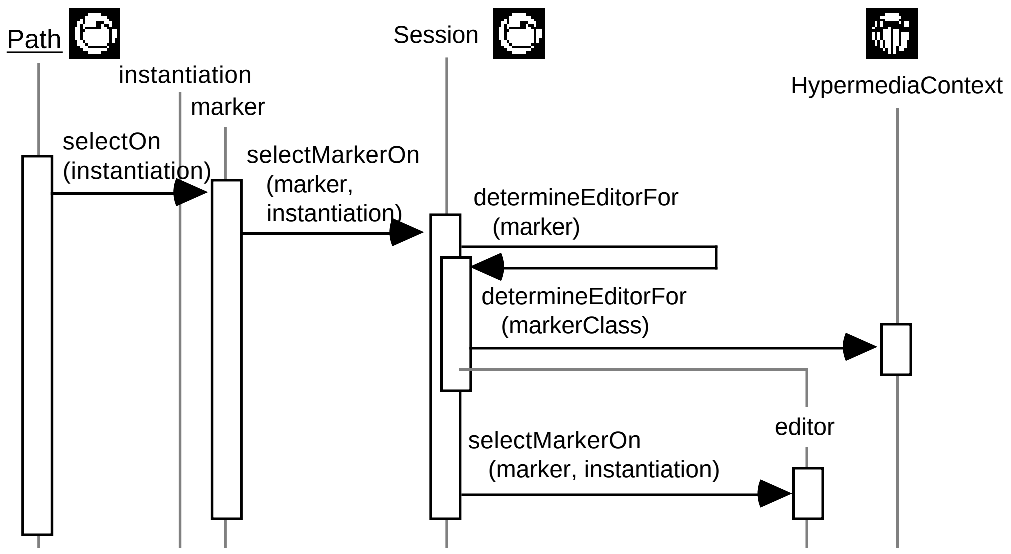

Path
Session
HypermediaContext
component
anchor
component PSpec
anchor PSpec
createFrom (component, componentPSpec)
interpretComponentAndSpec (component, componentPSpec)
determinePresentationClassFor (componentClass, componentPSpec)
instantiation
createFrom (anchor, anchorPSpec)
interpretAnchorAndPSpec (anchor, anchorPSpec)
determinePresentationClassFor (anchorClass, anchorPSpec)
marker

Having created objects representing target locations, the Path uses the rest of the quadruples to turn them into presentable objects. For each component (anchor), the Path object asks the Session meta-object to create an instantiation (marker) by sending the `createFrom` message. The Session object interprets that specifier with a self send of `interpretComponentAndPSpec` (`interpretAnchorAndPSpec`), which in turn requests the HypermediaContext object for the class to instantiate by means of the `determinePresentationClassFor`.

Step 6: Target presentation


Path
Session
HypermediaContext
instantiation
edit ()
editInstantiation (instantiation)
determineEditorFor (instantiation)
determineEditorFor (instantiationClass)
editor
editInstantiation (instantiation)
enumerateMarkers ()
marker
higlightOn (instantiation)

Having created the target instantiation object, the Path requests to present it by sending the `edit` message. On receiving the `edit` message, the instantiation passes it on to the Session meta-object. The Session first identifies the editor that is supposed to handle the request by sending the self message `determineEditorFor`. As part of this identification process, the Session requests the HypermediaContext meta-meta-object for the name of an installed and active editor. Afterwards, the Session transfers control to the identified editor by means of the `editInstantiation` message. On returning the Session enumerates all markers associated with an instantiation (by sending the `enumerateMarkers` message to the instantiation object) and highlights them by sending the `highlightOn` message. Note that the `highlightOn` message is processed like the `selectOn` message.



Having created the target marker object, the Path requests to select it by sending the `selectOn` message. On receiving the `selectOn` message, the marker passes it on to the Session meta-object. The Session first identifies the editor that is supposed to handle the request by sending the self message `determineEditorFor`. As part of this identification process, the Session requests the HypermediaContext meta-meta-object for the name of an installed and active editor. Afterwards, the Session transfers control to the identified editor by means of the `selectMarkerOn` message.

Step 7: Back to step 1

After the editor has displayed the instantiation and highlighted the associated markers it goes in sleeping mode, waiting for an event that activates the following navigation action.

Issues

Singleton Meta-Objects

The navigation template is a specification for the control flow in the Zypher open hypermedia framework. We see that a large part of the internal communication in the framework is message passing between the base level objects and meta-level objects. An obvious question is of course "What about performance" ?

It is clear that extra meta-level communication is an overhead compared to systems without a meta-level, thus that a meta-level has a bad effect on the performance. One may argue that this is reasonable: a meta-level is an extra abstraction layer which makes a system more adaptable but that adaptability comes at a cost. Although this is a valuable argument, it is not satisfying because it does not show the whole picture.

Reviewing the diagrams in the navigation template, we see that all the meta-level objects are presented as global objects and that all important messages are funnelled through these global objects. Software engineers recognise this as a "bottle-neck" pattern and know that bottle-necks usually have a bad effect on system's performance.

Yet, there is no rule that says that a meta-level object should be one global object. For instance, if one wants to set-up a system for a distributed environment then it is no use to have a central meta-object that must be consulted for each important activity in the system. To reduce network communication, a better approach would be to replicate the meta-level objects on each client system. In fact, replication is one example that illustrates the value of having a separate meta-level.

Consider a tele-presence situation, where two people are simultaneously editing a shared document and want to be aware of the modifications made by the other. Tele-presence can be achieved by adapting the editor software to pass all open, close and change events to the other application. Such an adaptation implies that one must change the internal implementation of the editor software to incorporate the message passing mechanism. If we review the Zypher design, we see that there is an event-passing mechanism available (see [events]) and that this event-passing is controlled by the meta-level objects (see [meta-objects], [meta-meta-objects]). This architecture allows to replicate the meta-level objects over the two machines and it is feasible to plug in special meta-objects that pass open- and close-events to all other meta-objects residing on other machines. This way, meta-level objects can detect other meta-level objects controlling the same document and synchronise with them by passing change-events. The base level objects remain unchanged, since the event-passing mechanism is still the same.

We conclude that a meta-level would allow to implement the tele-presence example without modifying the internal implementation of the editor software. Although we did not perform the actual experiment, other work has shown that this is feasible approach. CodA [McAffer'95] has been used to open up the implementation of Smalltalk message passing and is able to add meta-level infrastructure to Smalltalk objects so that additional behaviour like concurrency or distribution can be added when necessary. The CodA experiment is especially important as it shows that a meta-level is "just another application" and that traditional software design techniques like abstraction and decomposition remain valuable. Abstract Communication Types [AksitEtAl'93] are objects that abstract interactions among objects and can be used to model distribution and concurrency. Unlike CodA, the abstract communication types are intended to be part of the design of a new programming language (i.e. Sina). Apertos [Yokote,Teraoka,Tokoro'89], [Yokote'92] is object-oriented operating system for mobile computing, supporting distribution and object migration by means of its meta-object protocol. SOM, the System Object Model part of IBM's OS/2 operating system [Forman,Danforth,Madduri'95] includes the notion of meta-classes, which can be used to wrap additional behaviour (locking, notification) around base level message sends.

Relations

Where To Go Next ?

People that read the Zypher design pattern documentation sequentially came at the end. The obvious next step is to review the design by following a non-sequential path.

Other Catalogues

The Path, Session and Hypertext meta-objects and the HypermediaContext meta-meta-object are singletons [GammaEtAl'93]. This allows to implement them as global variables, accessible from the whole system, yet with a restricted access.