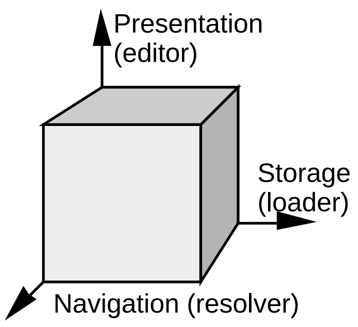
CHAPTER **5**

A Design Space For A Hypermedia Framework



This chapter introduces three behavioural objects in the design of the Zypher open hypermedia framework: resolver, editor, and loader.

The introduction of the behavioural objects is the first step towards modelling hypermedia systems using the Zypher design space (p.30). This design space is delineated by three orthogonal design axes, where each axis represents aspects important for all hypermedia systems, namely navigation, presentation and storage. A hypermedia system is then a three dimensional volume in that design space.

Based on the second of our framework design guidelines, stating that "Each design space axis should correspond to a black-box template method" we introduce a behavioural object to represent a point on one of these design axes (i.e. resolver for the navigation axis, editor for the presentation axis and loader for the storage axis). This allows us to incorporate the notion of domain level tailorability into the design of the framework.

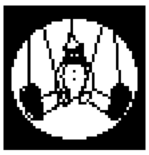
Behavioural Objects

Behaviour is the central theme underneath the design patterns in this chapter. All the design patterns listed here, describe objects representing behaviour for Zypher data structures. Behavioural objects are an internal representation for (possible third party) software that is supposed to store, present or navigate the information structures.

The design patterns in this chapter are quite similar. This should not come as a surprise as the same basic technique is at work here: to model a certain design space axis one favours the black-box approach over the white-box approach.

Resolver: The Core of an Extensible Link Engine

Intent



Encapsulate the algorithms that determine the navigation relations (i.e. what targets can be reached from a given source) to accomplish a hypermedia system with an extensible link engine.

Analysis

Navigation is an answer to the problem of seamless integration, but the solution introduces a new problem: how to manage the connections between related information ? Traditional hypermedia systems use static links for these purposes, which means that the author of a hypermedia system must maintain the connections between sources and targets of navigation relations manually. Static link creation is usually accomplished by some kind of recording mechanism, where users select parts of information fragments in viewer applications and issue a command to mark the selection as the source or target of a link.

Static links have the advantage of being applicable in all kinds of information structures, since they are independent of the problem domain: it is the author of a link that determines the relations between the information fragments and provides the semantics. However, maintenance of hypermedia systems structured with static links is a nightmare that makes the technique unworkable in dynamic environments.

An extensible link engine is another approach to manage the navigation relations. The link engine is the part of the hypermedia system that determines what targets can be reached from a given source. An extensible link engine is a link engine that is able to incorporate domain specific algorithms to infer relationships between information fragments. With an extensible link engine, the hypermedia system is able to incorporate knowledge about the information domain and to help in authoring and maintaining the navigation relations.

EXAMPLE

Hypermedia technology has been successfully applied in many technical documentation projects, so it should not come as a surprise to recommend a hypermedia system as an ideal medium to explore framework structures. However, compared with traditional technical documentation projects, framework structures demands an additional requirement from the underlying hypermedia system: flexible adaptation towards changes. A concrete framework applied in real life situations is constantly evolving. On the one hand, the framework is incarnated in applications part of the framework's target design space. On the other hand, the framework is redesigned to cope with new requirements. Both have impact on the framework structure and the hypermedia system should handle such structural changes effectively. To provide adequate support, the hypermedia system must incorporate knowledge from the programming environment to infer information structures automatically.

As an example, consider the links connecting a method to all senders (implementors) of the corresponding message. One cannot expect a software engineer to maintain such links manually: they should be inferred from the compiler part of the framework development environment. Moreover, to offer true support in managing frameworks, the links connecting a framework element to the design patterns it participates in should be maintained within the framework development environment as well.

Problem

How can one incorporate domain specific link resolution algorithms in the previously defined navigation model (see [navigation]), so that resolution algorithms can be maintained easily ?

Solution

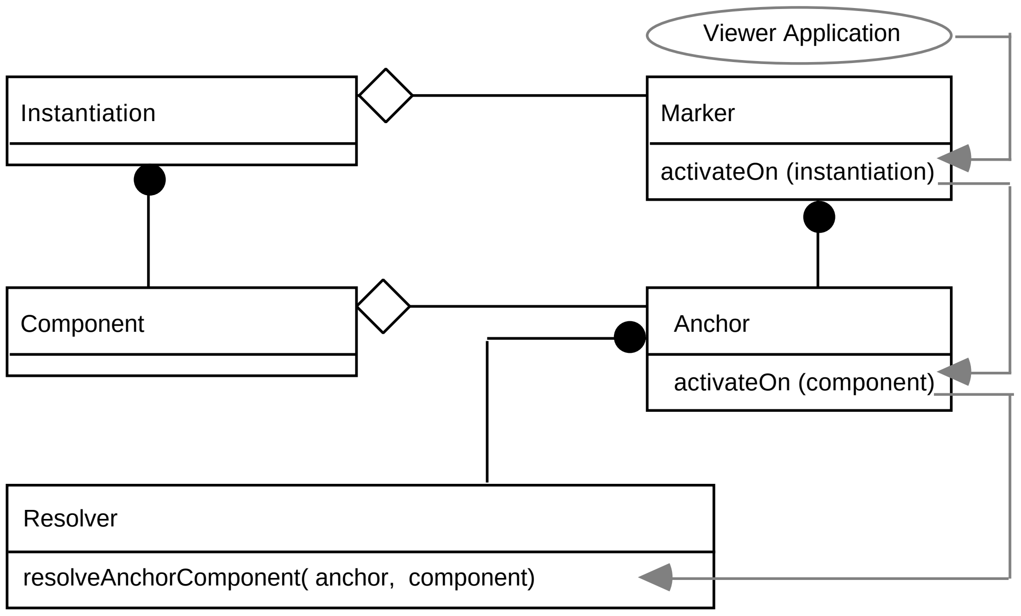
Create a new class of objects, where each object represents a link resolution algorithm. Call such an object a *resolver*. Define a contract on markers and anchors, stating that all navigation actions are passed to such a resolver to determine the appropriate targets.

EXAMPLE

The Zypher framework browser incorporates navigation actions that bring a software engineer from a given message selector to all the places where such a message is implemented and all the places where such a message is sent. This counts for two resolvers: a senders resolver and an implementors resolver. Also there are navigation actions that bring a software engineer from a framework element (a class or a method) to a design pattern where this framework element plays a particular role. Determining these target patterns is accomplished by the pattern resolver.

The Zypher framework browser also incorporates a resolver for every URL address space. Note that some special address spaces for the framework browser domain are added (browser, pattern, name).

Contract



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

Component

A component passed as a parameter to the `activateOn` method defined on Anchor.

Instantiation

An instantiation is passed as a parameter to the `activateOn` method defined on Marker.

Marker

All markers implement the `activateOn` message sent by the viewer application to start a navigation action. Markers receiving this message, must pass it on to the anchor they represent, but are allowed to perform additional actions.

Anchor

All anchors implement the `activateOn` message sent by a marker to start a navigation action. Anchors receiving this message, must pass it on to the resolver by means of the `resolveAnchorComponent` message, but are allowed to perform additional actions.

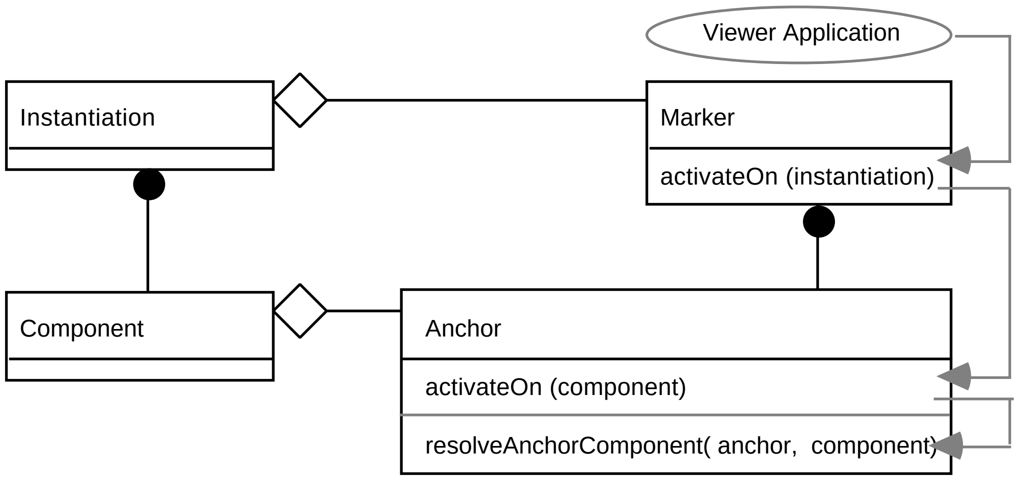
Resolver

A resolver models a specific link resolution algorithm. A resolver implements the `resolveAnchorComponent` message sent by the anchor to determine the target of a navigation action. Specific resolver classes are allowed to extend the message set for a finer grained interaction with the anchor.

Motivation

The determination of the navigation targets is crucial in all hypermedia systems. The need to incorporate domain specific knowledge forces us to make the determination process explicit in a resolver method. Why should this method be defined on a separate class, why not use an existing one ?

The most obvious candidate would be the Anchor class, which would result in the following structure.



In the above schema, an anchor captures two different aspects of the design, namely the structural reference to a substructure of a component and the way to resolve an anchor-component pair. Mixing two different aspects in a single design element causes difficulties as the design is less configurable and less extensible.

a) Configurability

Some anchors may resolve differently depending on the navigation context, which means that in some occasions the resolver process must change at run-time. Changing the resolver process is not possible if the two are captured in a single design element. Moreover, decoupling the resolver from the anchor enables lazy structures, one can defer to choice of the resolution algorithm until it is needed.

EXAMPLE

The Zypher framework browser incorporates navigation actions that bring a software engineer from a given message selector to all the places where such a message is implemented and all the places where such a message is sent. Thus the same anchor (the message selector) is used in different resolution algorithms: one wants to attach the resolver to the anchor depending on the particular action that triggers the navigation.

Zypher framework documentation is stored inside HTML-documents and Microsoft Word documents and contain many anchors in URL format. In the URL format, the anchor value starts with a keyword defining the URL address space, followed by some address in a format depending on the address space. The keyword identifies the resolver that is supposed to interpret the address. However, it is not a very good idea to parse and interpret all keywords in all anchors to determine the resolver. Most of the anchors available in the document are not activated, so it is better to wait until the actual activation to interpret the keyword.

b) Extensibility

Like argued in the [interoperability] pattern, separating the reference aspect from the resolver aspect reduces the number of classes by factoring out the behaviour along the two dimensions.

EXAMPLE

Message selectors and class names can be found in lists (i.e. the list panes of the pattern browsers and code browser) in which case an index anchor is used, or they can be found in Smalltalk source code in which case a range anchor is used. On both classes of anchors the senders, implementors and pattern resolver may be applied.

Both HTML-documents and Microsoft Word documents contain many anchors in URL format, referring to various address spaces. Anchors in a HTML-document hold a range to refer inside the document, while anchors for Microsoft Word documents use bookmarks (i.e. value markers).

As can be deduced from the table in [figure 19], we reduce the number of classes to twelve (= 9 + 3) instead of eighteen (= (3 * 2) + (6 * 2)). Moreover, all URL resolver classes are defined as a subclass from a common abstract URL resolver, which makes it very easy to extend the system with new URL address spaces.

Anchor \ Resolver	Index	Range	Value
Senders	x	x	
Implementors	x	x	
Pattern	x	x	
URL		x	x
URL File		x	x
URL HTTP		x	x
URL Browser		x	x
URL Pattern		x	x
URL Name		x	x

Figure 19: Combinations of anchors and resolvers

Issues

Naming

The concept of a resolver as defined here is not part of the Dexter model [Halasz,Schwartz'90]. However, the Dexter specification includes the notion of a resolver function (Dexter is quite vague there), which explains the choice of the name.

Structural Objects versus Behavioural Objects

It was not a coincidence that the same arguments (i.e. configurability and extensibility) that plead for a separation between components, anchors, instantiations and markers, motivate the existence of a separate resolver object. However, the absence of the dynamics argument, suggests that there is another factorisation technique at work here.

The difference between the factorisation applied in the [interoperability] and [navigation] patterns and the factorisation applied here is based on the difference between structure and behaviour.

To understand why, note that the component, anchor, instantiation and marker objects represent elements for data structures: they must be assembled properly to model external information. The resolver object, however, represents a process that interprets a particular constellation of these elements. The resolver thus assigns behaviour to a structure, which explains the terms structural object and behavioural object.

Consequences

If the configuration issue was important when discussing structural objects (see [interoperability ~ consequences], [navigation ~ consequences]), then it is even more important if a design incorporates behavioural objects. After all, the decision about what behavioural object should handle what structural objects has a great impact on the global behaviour of the system. The issue is dealt with in the [meta-meta-objects] design pattern.

Relations

Where To Go Next ?

People reading the Zypher design pattern documentation for the first time should have learned how the introduction of the resolver object allows to incorporate domain specific algorithms that determine navigation relations. The obvious next step is then the [editor] pattern, that models a viewer application.

The [meta-meta-objects] pattern determines what resolver should handle a given anchor/component pair.

The resolver plays an important role in the navigation template as described in the [navigation template].

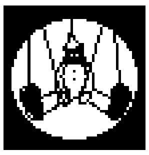
Other Catalogues

The messages defined in the contracts section participate in the template method pattern as defined in [GammaEtAl'93] and [Pree'94]. To use the terminology of the latter, the `resolveAnchorComponent` method defined on resolver is a hook method for the `activateOn` template method defined on anchor. In turn, the `activateOn` method defined on anchor is a hook method for the `activateOn` template method defined on marker. Both are forms of the 1:1 connection meta-pattern. The `activateOn` and `resolveAnchorComponent` messages are hook methods for the navigation template [navigation template].

Besides participating in the 1:1 connection meta-pattern, the `activateOn` messages defined on anchor and marker participate in the 1:1 recursive connection meta-pattern, meaning that subclasses overriding the `valueOn` message must include the super method in their behaviour.

Editor: Incorporate External Viewer Applications

Intent



Encapsulate the command sets that control the viewer applications presenting information and participating in the navigation.

Analysis

Hypermedia is about managing and presenting information. A crucial part of all hypermedia systems is the module that is responsible for presenting information. Presenting information in a hypermedia system is concerned with presenting a particular information fragment, highlighting substructures of an information fragment as candidate source of a navigation operation and selecting a substructure of an information fragment as target of a navigation operation.

A complicating factor is the notion of [interoperability], where the hypermedia system is able to incorporate third party applications to present information to the user and third party applications can request link services from the hypermedia system. Moreover, the same information can be presented by different viewer applications. Note that these applications are allowed to modify both the information and the navigation structures.

Problem

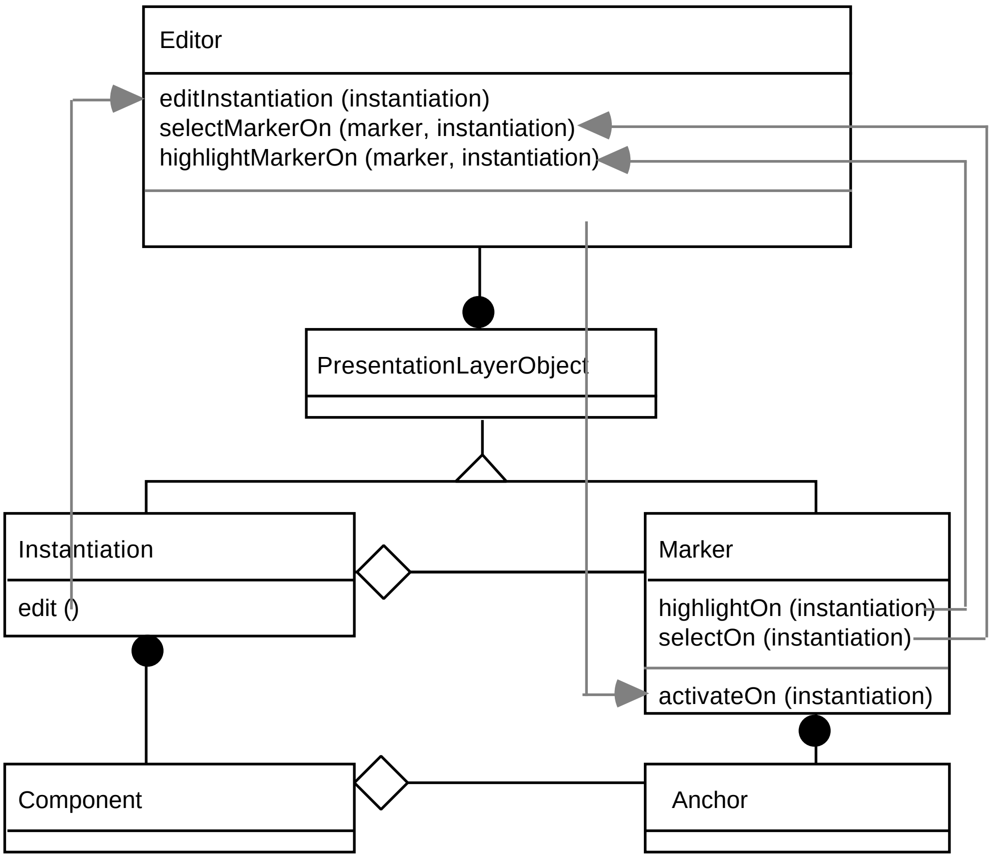
How can one model an interoperable information system where the same information can be manipulated by different presentation applications, possibly not part of the hypermedia system ?

Solution

Create a special object that represents an information presentation application. Call this object an *editor*, to emphasise the idea that these applications can modify information.

Create a special superclass for all objects that can be manipulated by an editor object; call this a *PresentationLayerObject.*

Contract


Editor
editInstantiation (instantiation)
selectMarkerOn (marker, instantiation)
highlightMarkerOn (marker, instantiation)
PresentationLayerObject
Instantiation
edit ()
Marker
highlightOn (instantiation)
selectOn (instantiation)
activateOn (instantiation)
Component
Anchor

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

Editor

An editor object represents a module of the hypermedia system (possibly an interface to a third party application) that can be asked to present information.

All editors implement the `editInstantiation` method called by an instantiation object that wants to be presented. All editors implement the `highlightMarker` method called by a marker object that wants to be highlighted as a candidate source for a navigation operation. All editors implement the `selectMarker` method called by a marker object that wants to be selected as a target for a navigation operation.

PresentationLayerObject

A presentation layer object is an abstract superclass for all objects that can be manipulated by an editor object.

Instantiation

All instantiations implement the `edit` method called by whatever object that wants to present the instantiation. This method is delegated to an `editInstantiation` method defined on an editor associated with the instantiation.

Marker

All markers implement the `highlightOn` method called by whatever object that wants to highlight the marker as a source of a navigation operation. This method is delegated to an `highlightMarkerOn` method defined on an editor associated with the marker.

All markers implement the `selectOn` method called by whatever object that wants to select the marker as a target of a navigation operation. This method is delegated to an `selectMarkerOn` method defined on an editor associated with the marker.

All markers implement the `activateOn` method called by an editor object when it detects that the user has activated this marker as a source of a navigation operation.

Motivation

Why should the `edit, highlightOn, selectOn` methods be defined on a separate class. The same arguments that made us separate the resolver from the anchor (see [resolver]) apply (i.e. configurability and extensibility).

Consequences

If the configuration issue was important when discussing structural objects (see [interoperability ~ consequences], [navigation ~ consequences]), then it is even more important if a design incorporates behavioural objects. After all, the decision about what behavioural object should deals with what structural objects has a great impact on the global behaviour of the system. The issue is dealt with in the [meta-meta-objects] design pattern.

Relations

Where To Go Next ?

People reading the Zypher design pattern documentation for the first time should have learned how the introduction of the editor object allows to incorporate specialised viewer application. The obvious next step is then the [loader] pattern, that models an information repository.

The [meta-meta-objects] pattern determines what editor should handle a given instantiation or marker.

The editor plays an important role in the navigation template as described in the [navigation template].

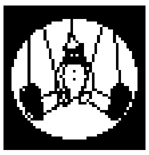
Other Catalogues

The messages defined in the contracts section participate in the template method pattern as defined in [GammaEtAl'93] and [Pree'94]. To use the terminology of the latter, the `editInstantiation, selectMarkerOn, highlightMarkerOn` methods defined on editor are hook methods for the `edit` template method defined on instantiation and the `highlightOn`, `selectOn` template methods defined on marker. In turn, the `activateOn` method defined on marker is a hook method for an unspecified template method defined on editor. Both are forms of the 1:1 connection meta-pattern. The `edit, highlightOn` and `selectOn` messages are hook methods for the navigation template [navigation template].

Loader: Incorporate Information Repositories

Intent

Encapsulate the commands that control the repositories storing the information.



Analysis

Hypermedia is about managing and presenting information. A crucial part of all hypermedia systems is the module that is responsible for storing and retrieving information from information repositories.

A complicating factor is the notion of [interoperability], where the hypermedia system must be able to incorporate various information repositories outside the hypermedia system. This implies that the information can be changed from outside the hypermedia system.

Problem

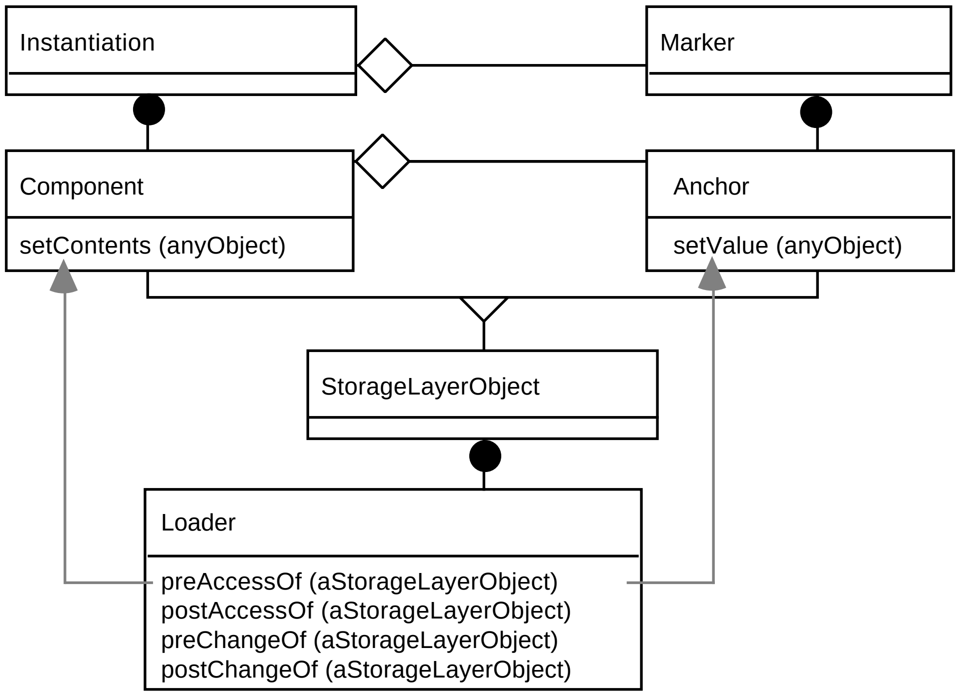
How can one model an interoperable information system where information can be supplied from different repositories, not part of the hypermedia system ? How can one deal with the problem that information can be modified from outside as from within the system ?

Solution

Create a special object that represents an information repository. Call this object a *loader*.

Create a special superclass for all objects that can be manipulated by a loader object; call this a *StorageLayerObject*.

Contract


Instantiation
Marker
Component
setContents (anyObject)
Anchor
setValue (anyObject)
StorageLayerObject
Loader
preAccessOf (aStorageLayerObject)
postAccessOf (aStorageLayerObject)
preChangeOf (aStorageLayerObject)
postChangeOf (aStorageLayerObject)

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

Loader

A loader object represents a module of the hypermedia system (possibly an interface to a third party application) that can be asked to store or retrieve information.

All loaders implement the `preAccessOf`, `preChangeOf`, `postAccessOf`, `postChangeOf` messages, sent whatever object that accesses or changes a storage layer object. The loader then checks whether the contents of the storage layer object corresponds with what is in the repository and adapts the contents when necessary.

StorageLayerObject

A Storage layer object is an abstract superclass for all objects that can be manipulated by an editor object.

Component

All components implement the `setContents` method called by the loader when it detects that the contents of a component does not correspond with the contents in the repository.

Anchor

All anchors implement the `setValue` method called by the loader when it detects that the contents of an anchor does not correspond with the contents in the repository.

Motivation

Why should the `preAccessOf`, `preChangeOf`, `postAccessOf`, `postChangeOf` methods be defined on a separate class. The same arguments that made us separate the resolver from the anchor (see [resolver]) apply (i.e. configurability and extensibility).

Relations

Where To Go Next ?

People reading the Zypher design pattern documentation for the first time should have learned how the introduction of the loader object allows to incorporate specialised repositories. The obvious next step is then the [evens] pattern, that imposes a layered structure on top of the Zypher framework and introduces event passing as a mechanism to transfer results from lower layers to upper layers.

The [meta-meta-objects] pattern determines what loader should handle a given component or anchor.

The loader plays an important role in the navigation template as described in the [navigation template].

Other Catalogues

The messages defined in the contracts section participate in the template method pattern as defined in [GammaEtAl'93] and [Pree'94]. To use the terminology of the latter, the `setContents` method defined on component and the `setValue` method defined on anchor is a hook method for an unspecified template method defined on loader. This is a form of the 1:1 connection meta-pattern.