Vrije Universiteit Brussel
Faculteit Wetenschappen

The Zypher Meta Object Protocol

Serge Demeyer - Koen De Hondt - Patrick Steyaert
- Wim Codenie - Roel Wuyts - Theo D'Hondt

Presented at the second workshop on open hypermedia systems (Washington DC, US — 1996)
Published in Wiil, U. K. / Demeyer, S. (Ed) "Proceedings of the 2nd Workshop on Open Hypermedia Systems - Hypertext'96"; UCI-ICS Technical Report 96-10. Department of Information and Computer Science, University of California, Irvine, CA 92717-3425.

Techreport vub-prog-tr-96-04

Programming Technology Lab
PROG(WE)
VUB
Pleinlaan 2
1050 Brussel
BELGIUM

Fax: (+32) 2-629-3525
Tel: (+32) 2-629-3308
Anon. FTP: progftp.vub.ac.be
WWW: progwww.vub.ac.be

The Zypher Meta Object Protocol

POSITION STATEMENT SUBMITTED TO
THE 2ND WORKSHOP ON OPEN HYPERMEDIA SYSTEMS
Hypertext '96 Conference - March '96 - Washington (US)
http://www.iesd.auc.dk/~kock/OHS-HT96/ - http://www.acm.org/siglink/ht96/

Serge Demeyer
Patrick Steyaert - Koen De Hondt - Wim Codenie - Roel Wuyts - Theo D'Hondt
Vrije Universiteit Brussel / Faculty of Sciences
Programming Technology Lab (PROG) Pleinlaan 2
B-1050 Brussels (Belgium)
phone: (+32) 2 629 34 91
{sademeye | prsteyae | kdehondt | wcodenie | tjdhondt}@vnet3.vub.ac.be; rwuyts@is1.vub.ac.be
http://progwww.vub.ac.be/

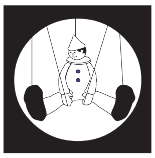
Abstract

This paper discusses the necessity of a meta object protocol in the design of an open hypermedia system. It shows that a meta object protocol enables to tailor the behaviour and configuration of the hypermedia system, independent of its constituting elements.

The approach is demonstrated by means of the Zypher Open Hypermedia Framework, where the meta object protocol eases the incorporation of system services (i.e. caching, logging, authority control and integrity control) and flexible reconfiguration (i.e. run-time extensibility and cross-platform portability).

1. Introduction

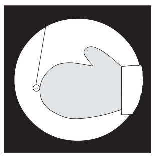
To understand the argumentation unfolded in the main body of the paper, it is necessary to emphasise that Zypher[1] was explicitly designed as an open hypermedia system with three levels of tailorability. Each level provides different facilities to suit the behaviour of the system to the needs of particular hypermedia applications.



Domain Level

Domain level tailorability aims to deliver hypermedia systems for a specific problem domain by extending the basic hypermedia framework with domain specific modules. Creating such domain specific modules requires a great deal of technical expertise about the software systems applied in the problem domain but has little to do with the hypermedia system as such. One doesn't need to understand the inner details of the hypermedia system to tailor the system. Note that some modules, if written 'good', can be reused for different problem domains.

Typical usage of domain level tailorability is the incorporation of modules for special viewer applications (i.e. Microsoft Word, a HTML browser), extra storage devices (i.e. the local file system, a HTTP-server) and designated navigation facilities (i.e. special URL resolution algorithms).



System Level

System level tailorability aims to deliver services that affect the global behaviour of the hypermedia system itself and requires some knowledge about the internal architecture of the hypermedia system. Services attained trough system level tailorability can be applied on different incarnations of the hypermedia framework: once we have implemented the technique in one framework incarnation, it requires little effort to reuse the code in other incarnations.

Typical examples of services that can be accomplished with system level tailorability are things like logging (maintaining a log of certain activities in order to provide backtracking features), authority control (check whether the user of the system has the privileges to perform certain operations), caching (predict future behaviour on the basis of registered activities) and integrity control (control operations in order to preserve the consistency of the system's data structures).

1 The name Zypher stems from the Louis Zypher character performed by Robert De Niro in the movie "Angel Heart".

Configuration Level

Configuration level tailorability aims to provide a 'plug and play' hypermedia system, where the system configuration is adapted without modifying the constituting modules. This accommodates for a flexible system set-up, where new modules can be installed easily. Configuration level tailorability requires a deep knowledge about the internal architecture of the hypermedia system; however technical details about individual modules do not matter.

Typical examples of configuration level achievements are flexible configuration (i.e. run-time extensions to the system) and portability (i.e. cross-platform reconfiguration).

These levels of tailorability are quite important and hypermedia system designers will often need to switch between these levels in order to develop a particular hypermedia application. That is why we have devised special icons that are employed in the framework documentation and throughout the remainder of this text[2]. The icons are based on the *puppet master metaphor* (see [figure 1]).

- When preparing a story, the puppet designer will conceive a number of puppets playing different characters. To distinguish these characters the puppets will be dressed with different costumes and their faces will be painted. Typical puppet characters are the harlequin and the pierrot, the former wearing a costume with lots of coloured patchwork and a smiling face, the latter is dressed in white with a tear under the eye. This kind of tailorability corresponds with the domain level tailorability and is visualised using an icon presenting a puppet.
- However, for certain kinds of stories, some puppets require special abilities that demand for extra strings to manipulate the special behaviour. Some scene in the play might benefit from a horse with a swinging tail, in which case the puppet designer will take an existing horse puppet and attach a new string to the tail. A puppet designer that attaches new strings to puppets is a designer that operates on the system level of tailorability, which is denoted by means of a hand-with-string icon.
- Finally, the way the strings work together is implemented in the wooden cross manipulated by the puppet player. A puppet designer creating a knight on a horse fighting with a spear will adapt the branches of the wooden cross to operate the puppet and works on the configuration level of tailorability. This is symbolised with a cross icon.

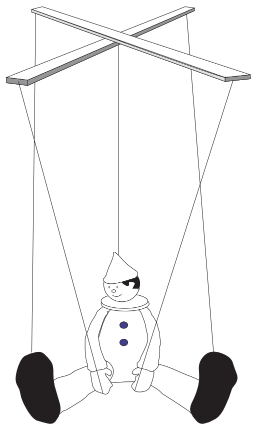


Figure 1: Puppet Master Metaphor

(*remark:* Note that the users of the hypermedia system correspond with the audience watching the puppet: they are not supposed to know how the puppet is manipulated to produce the desired scenes in the performance. However, just like the audience can influence the play by applauding and shouting, users can influence the behaviour of the system by setting preferences. Actually, the puppet master —i.e. the hypermedia system designer— will use the appropriate level of tailorability to satisfy the audience).

Document Organisation

The remainder of this paper will be used to demonstrate how techniques from the object-oriented software engineering community may help to develop and maintain a hypermedia system with the three levels of tailorability. More precisely, the aim of the paper is to show that the introduction of a *meta object protocol* delivers the desired system level and configuration level tailorability (see sections 3 and 4). Before discussing the notion of a meta object protocol, we will discuss the issue of domain level tailorability (see section 2), where we will define an *object-oriented framework* for the domain of Open Hypermedia Systems.

2. The Base Level

This section will settle the scope of the rest of the paper, with a design specification for the Zypher Open Hypermedia Framework. It is important to note that this specification is not complete and this for three reasons. First of all, the design will be gradually improved (by adding meta objects) when we introduce the notions of system level and configuration level tailorability. Secondly, because the design of the base level is not crucially important for the real issue: the necessity of a meta object protocol). All that really matters is that there exists a base level design, and that it is formalised in a set of contracts specifying relations between objects (we refer the interested reader to [Demeyer'96] for a full specification in design pattern form). Thirdly —and this follows from the previous motive— because it is possible to introduce a meta object protocol on any system with a base level design based on an object-oriented framework. The third point is extremely important, as it makes the technique of a meta object protocol applicable in many other hypermedia systems.

2 The idea of visualising the levels of the system by means of icons is adapted from [Kiczalis,Rivières,Bobrow'91].

A Design Specification

The design of the Zypher framework was based on the Dexter model [Halasz,Schwartz'90], well known in the Hypermedia community. Zypher retained the separation between the storage layer and the run-time layer (called presentation layer in Zypher) and the main elements of the Dexter factorisation (i.e. component, anchor, instantiation and marker). To handle the specific problems of an 'Open' hypermedia model[3], we extended the model with elements that represent a viewer application (i.e. editor), an information repository (i.e. loader) and a link resolution algorithm (i.e. a resolver) . This resulted in the object model (using OMT notation; see [Blaha,Premerlandi,Rumbaugh'88] and [RumbaugEtAl'91]) depicted in [figure 2].

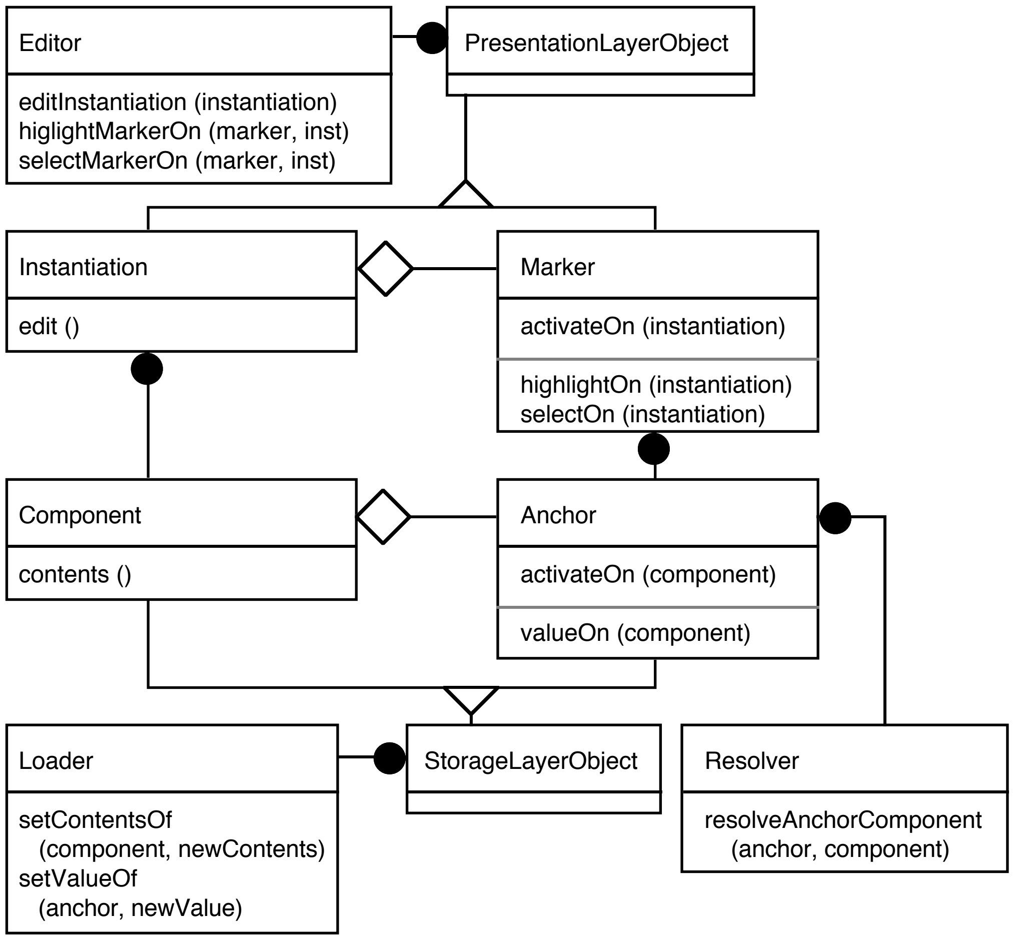

Editor
editInstantiation (instantiation)
higlightMarkerOn (marker, inst)
selectMarkerOn (marker, inst)
PresentationLayerObject
Instantiation
edit ()
Marker
activateOn (instantiation)
highlightOn (instantiation)
selectOn (instantiation)
Component
contents ()
Anchor
activateOn (component)
valueOn (component)
Loader
setContentsOf (component, newContents)
setValueOf (anchor, newValue)
StorageLayerObject
Resolver
resolveAnchorComponent (anchor, component)

Figure 2: The Design of the Base Level

Instead of presenting an explicit enumeration of all contracts, that exist between the objects defined in [figure 2], we will give a brief description of the message flow that implements the navigation operation (the heart of all hypermedia models).

Step 1: Selection of navigation source

An instantiation represents a document as it is displayed by some viewer application (the viewer application is represented by the editor object). An instantiation contains a number of markers (representations for the visible sources or targets for navigation operations) which explains the aggregation relation between instantiation and marker.

To start a navigation action, the editor will send an #activateOn message to a marker with the associated instantiation as parameter. The marker is allowed to produce some visual effects before proceeding with the next step.

Step 2: Identification of navigation source

The 1-to-many associations between instantiation/component and marker/anchor will be used to find the associated anchor-component pair. This pair identifies the source of the navigation operation and the marker must send the #activateOn message on to the associated anchor supplying as parameter the component associated with the instantiation.

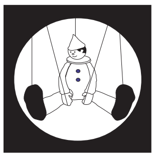
3 See [Demeyer'96] for a motivation of these extensions.

Step 3: Resolution Process

The 1-to-many association between anchor/resolver will be used to retrieve the resolver object containing the algorithm that produces the target of the navigation action. The #resolveAnchorComponent message must be send to this resolver to obtain a collection of quadruples where each quadruple represents one target of a navigation action. Inspired by the Dexter model, each quadruple contains a component specifier, a component presentation specifier, an anchor specifier and an anchor presentation specifier; such quadruple will be turned into a new component, anchor, instantiation and marker that will serve as the target of the navigation operation.

Step 4: Target Presentation

For each target the message #edit will be sent to the instantiation; the instantiation must pass this message on the associated editor (by means of the #editInstantiation message) to instruct the viewer application to open a view. Afterwards, all associated markers will be sent the #highlightOn message which must be passed on to the associated editor (by means of the #highlightMarkerOn message) to instruct the viewer application to highlight them as candidate sources or targets for future navigation actions. Finally, the actual target of the navigation action will be selected by sending the #selectOn message to the target marker, which must be passed on to the associated editor (by means of the #selectMarkerOn message). During this process, it is always possible to request a component for its contents (an anchor for its value) by sending the #contents (#valueOn) message. For un-initialised components (or anchors) the associated loader will supply the actual contents (value) using the #setContentsOf (#setValueOf) message.

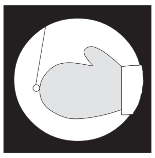


To verify the notion of domain level tailorability, we extended Zypher with several modules for the so-called "framework browser" problem domain. In order to explain the framework browser concept, we must clarify some related concepts. *Object-oriented frameworks* are the state of the art in object-oriented software engineering and consist of a tight co-operation of the analysis, design and implementation concepts modelling a particular application domain. A framework consists of different *design patterns* (see [Johnson'92], [Beck,Johnson'94], [GammaEtAl'93], [Pree'94]) that focus on a single analysis, design and implementation aspect of the overall framework structure. A *framework browser* is then an integrated set of tools to manipulate the design patterns inside a framework. Currently, these tools are

- a home cooked HTML browser (used to read design pattern documentation),
- the Microsoft Word third party application (used to produce design pattern documentation),
- several code browsers (used to modify the implementation of the framework) and
- several pattern browsers (used to match the implementation expressed in concrete classes with the design specified in contracts between abstract classes).

The Zypher link engine seamlessly integrates all these tools by providing navigation facilities from one tool to another. For example, it is possible to follow hypermedia links from the design pattern documentation (i.e. a HTML or Microsoft Word document) to the implementation (i.e. a code or pattern browser). Also, one can make hyper jumps from the implementation to the design pattern documentation describing that part of the framework.

3. The Meta Level



To verify the notion of system level tailorability, we decided to experiment with a backtrack function for all the navigation actions performed by the hypermedia system. A backtrack function is often helpful in hypermedia systems, as it is one of the techniques to handle the well known 'lost in hyper space' phenomenon (i.e. [Zellweger'89] and [Conklin'87]).

Keeping track of all navigation actions boils down to the maintenance of a log: for all navigation actions we must save the internal state of the participating agents to be able to restore them later. From this insight follows that an implementation must solve two problems in order to provide a working backtracking service. There is the problem how to ensure that all navigation actions are witnessed and there is the problem how to save the internal state of the participants. The former problem will be discussed in the following section; the latter is beyond the scope of this paper. Briefly we can say that the introduction of special state objects memorising the internal state of an object solves the problem. The technique is based on the 'Memento' design pattern, as described in [GammaEtAl'93]; we refer the interested reader to [Demeyer'96] for a more detailed description.

Funnel Navigation Actions

Re-examining the base level design of the Zypher system (see [figure 2]), we find that the navigation operation is modelled with a few key messages defined on the participating agents (i.e. the #activateOn message defined on marker, the #activateOn message defined on anchor, or the #resolveAnchorComponent defined on resolver). If we want to log all navigation actions, this would imply a patch of all implementations of at least one of these key messages. From a software engineering perspective, this is an unfavourable situation as it causes redundancy: the implementation of the logging algorithm is duplicated over all implementations of the patched key message. From an open hypermedia perspective the situation is even worse, because in an extensible set-up, the objects participating in the navigation action may be supplied by external sources. This means that there is no secure way to incorporate the log algorithm in all implementations, which implies that we can not ensure the integrity of the log.

To ensure that all navigation operations are witnessed by the log algorithm we must adapt the design of the hypermedia system by providing a funnelling point for all navigation actions. In the Zypher design, such funnelling point is accomplished in the so-called 'path'[4], an object with the explicit responsibility to control all navigation operations. The adapted model is depicted in [figure 3] (to avoid a cluttered figure, we left out some of the objects and most of the messages).

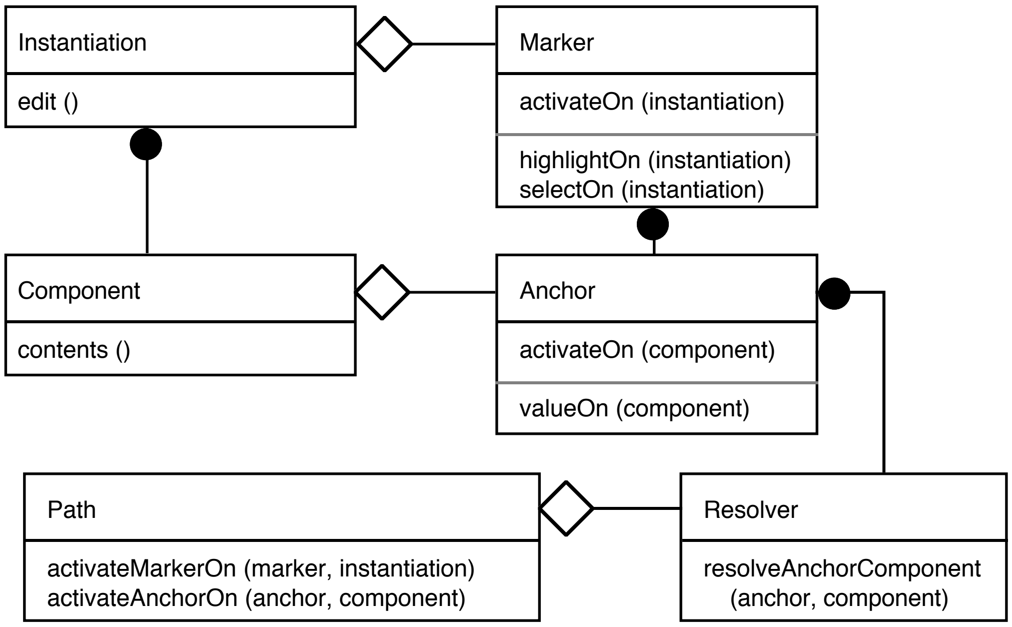

Instantiation
edit ()
Marker
activateOn (instantiation)
highlightOn (instantiation)
selectOn (instantiation)
Component
contents ()
Anchor
activateOn (component)
valueOn (component)
Path
activateMarkerOn (marker, instantiation)
activateAnchorOn (anchor, component)
Resolver
resolveAnchorComponent (anchor, component)

Figure 3: The Path Meta Object

The semantics of the adapted model (see [figure 3]) is as follows. There is exactly one path object for each hypermedia system. The implementation of the #activateOn message on all marker objects must delegate to the global path object implements by means of the #activateMarkerOn message; the implementation will perform the four steps involved in the navigation operation (i.e. selection of navigation source - identification of navigation source - resolution process - target presentation) by sending the appropriate messages to the participating objects.

Funnel Storage & Presentation Layer operations

Services like authority control, caching and integrity control have much in common with the logging example from above. They all depend on the ability to control all occurrences of particular messages being sent, regardless of the objects involved. For example, to implement authority control one wants to adapt all implementations of all #edit messages on all instantiations to check whether the user has the appropriate privileges; to maintain a cache of visited information one wants to patch all implementations of all #contents, #setContents, #valueOn and #setValueOf messages on all components and anchors; to ensure the integrity of the system's data structures one will control all operations that change the associations between the objects.

Like argued above, the best way to control all occurrences of a particular message is to provide a funnelling point. The design of the Zypher framework includes the 'session' object to funnel all presentation layer operations and the 'hypertext' object to funnel all storage layer operations[5]. The result is depicted in [figure 4].

[4] The name stems from the work of Zellweger [Zellweger'89].

[5] The Dexter model [Halasz,Schwartz'90] furnished the names 'hypertext' and 'session' because these objects are responsible for the management of the storage and presentation layer respectively.

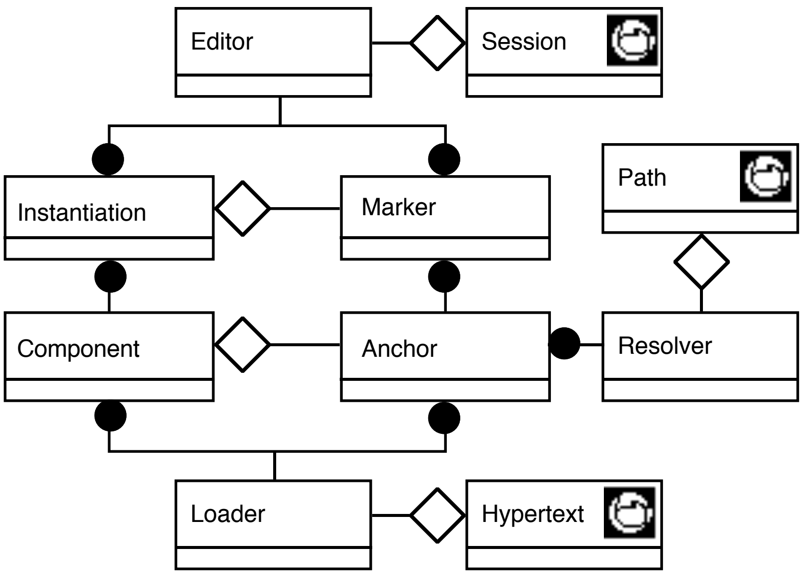


Figure 4: The Path, Hypertext and Session Meta Objects

Why Meta ?

In the previous sections, we have motivated the introduction of 'funnel' objects to provide system level tailorability. Now, we will argue why such funnel objects may be called meta objects. The argumentation relies on the fact that a meta objects is an explicit representation of contracts defined between objects in the base level design.

The term meta is generally connoted with the notion of *reflection*, i.e. the ability of a system to inspect and modify representations of it's own activities. Reflection is an intriguing idea —certainly within computer science— but is mostly considered an academic issue. Reflection has been studied in the area of artificial intelligence and the design of computer languages for quite a long time now (i.e. [Maes'87], [Kiczalis,Rivières,Bobrow'91], [Steyaert'94]). There, it has been shown that reflection eases extensibility (i.e. define a small and fixed kernel language and use that kernel to extend the language expressiveness), backward compatibility (i.e. compatibility with older definitions of the language) and efficiency (i.e. differ the implementation strategy to optimise behaviour). Moreover, since a reflective system is able to monitor its own activities, powerful tools like debuggers and code optimisers can be constructed more comfortably.

Recently the idea has been applied on the design of systems other than programming languages (i.e. [Rao'91]), leading to what has been called *implementational reflection* (or sometimes *open implementations*). A system with implementational reflection has the ability to inspect and/or change the implementational structures of its subsystems. Implementational reflection does not directly provide solutions for the problem domain the system has been designed for, but it does contribute to the internal organisation and the external interface of that system. This suggests that what we have been calling system level tailorability is indeed a feature that can be attained with implementational reflection.

To explain why the funnel objects make the hypermedia system a reflective one, we turn to the definitions found in [Maes'87]. There, *a reflective system is defined as a system which is about itself in a causally connected way*. We elaborate on the three main ideas in this definition (i.e. system, about-ness and causal connection) to make things more precise. A '*system*' is software running on a computer with the intention to answer questions about and/or support actions in some domain. A system will incorporate internal structures representing it's domain, that is why a system is said to be '*about*' it's domain. A system is said to be '*causally connected*' to its domain if the internal structures and the domain they represent are linked in such a way that if one of them changes, this leads to a corresponding effect on the other. In an object-oriented implementation of a system, the parts of the system that represent causally connected internal structures are called meta objects.

The definition of causal connection implies that a causally connected system may actually cause changes in the problem domain by a mere change in the internal representation of that problem domain. As a consequence (since a reflective system incorporates structures that are causally connected to itself) a reflective system can actually modify itself by changing its internal representation.

To argue why the funnel objects (i.e. path, session, hypertext; see [figure 3] and [figure 4]) defined in the previous sections are meta objects, we must prove that these objects are (a) about the hypermedia system in (b) a causally connected way. The proof follows from the insight that the funnel objects are *explicit representations of the contracts defined between the objects on the base level*. Indeed, the important messages are specified in the static part of the contracts (i.e. the interface of the different objects as shown in [figure 2]), thus part of the design. However, without the funnel objects, the dynamic part of the contracts (i.e. the decision when a certain message is sent) is delegated to the implementation and it is precisely the dynamic part of the contracts that determines the system's behaviour. If the design is extended with the specially created path, session and hypertext objects (see [figure 3] and [figure 4]) the dynamic parts of the contracts are explicitly available, since all occurrences of all important messages arrive at, or originate from such funnel objects. Knowing that the specially created path, session and hypertext are representations of the dynamic parts of the contracts between the base level objects —specifying how the system should behave under certain conditions— , they are by definition 'about' the system. Moreover, they are an explicit representation of the contracts: changing the implementation of a funnel object will have immediate effect on the subsequent behaviour of the system so we conclude that they are 'causally connected' to the system.

4. The Meta Meta Level

In the previous section we have introduced meta objects (i.e. path, session and hypertext) as the explicit representations of the contracts defined between base level objects (i.e. component, anchor, instantiation, marker, resolver, editor and loader). However, the introduction of meta objects leads to supplementary contracts, which raises the question whether it is worthwhile to make these supplementary contracts explicit as well.

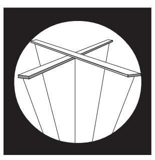
To show that it is worthwhile, this section will start with a summary of the contracts introduced by the meta level objects, followed by a description of two experiments with configuration level tailorability and ending with a discussion on the connection between meta meta objects and configuration level tailorability.

Meta Object Contracts

The meta objects are defined as objects controlling all operations concerning a particular layer (i.e. path for the navigation layer, session for the presentation layer and hypertext for the storage layer). As we can expect from this definition, there are more operations defined on meta objects, as the ones that follow from funnelling base level operations. A quick look at the design with the meta level objects (see [figure 4]) learns that the introduction of the meta objects adds operations for the aggregation relationships path-resolver, session-editor and hypertext-loader. The role of these aggregation relationships is to specify what kind of resolvers, loaders and editors are installed in the hypermedia system, which corresponds to the management of the available peripheral systems of the hypermedia system. Likewise (not visible in [figure 4]) the meta objects participate in aggregation relationships specifying the available classes (i.e. hypertext - component class; hypertext - anchor class; session - instantiation class; session marker class), which corresponds to the supervision of the potential elements constituting the running hypermedia system. Finally, (not depicted in [figure 4]), the meta objects introduce operations to create, query and release associations between objects (storage layer object - presentation layer object in session; instantiation - marker and component - anchor in path) plus relations between normal objects and peripheral objects (presentation layer object - editor in session; storage layer object - loader in hypertext; anchor - resolver in path). The role of these operations is to govern the connections between the internal elements of the hypermedia system and the link with the outside world.

The previous paragraph is a very short description of the contracts introduced by the meta level objects (for a detailed description we refer to [Demeyer'96]) which shows that —besides funnelling the navigation, presentation and storage layer operations— the meta objects do implement the configuration of the hypermedia system. This suggests that an explicit representation of the contracts defined on meta level objects may lead to the required configuration level tailorability.

Configuration Level Tailorability Experiments



To explore the notion of configuration level tailorability, we conducted two experiments. The first one is based on an interpretation of the URL (universal resource locator) format for anchors as defined in the HTML specification (see [Berners-LeeEtAl'94]). HTML documents embed their anchors in their documents using the URL format and we applied the same technique for the Microsoft Word documents. The URL format is open in the sense that it is prefixed by a keyword identifying the target address space, followed by an address in a format depending on the keyword prefix. The list of possible keywords is in principle unlimited, so the linking potential is only bounded by the list of interpretable keywords. In the Zypher hypermedia framework, the link engine consists of the list of resolvers installed in a path, so the mapping of the keyword-prefix on the appropriate resolver is the crucial process in the configuration of the hypermedia system. This mapping process is available by means of the #determineResolverFor message on the path meta object.

The second experiment has to do with the system's configuration across platforms. The Zypher framework documentation is organised as a collection of design patterns containing embedded anchors referring to related design patterns. The referencing is done by name, i.e. there is a special URL format starting with the keyword 'pattern' and followed by the name of the particular design pattern. The pattern resolver will turn this name into a file-name containing the design pattern document. However, the design pattern may come in a HTML and Microsoft Word version. The Microsoft Word version is richer (i.e. can be edited, contains pictures) but is only available on the Windows platform. On other platforms software engineers must use the HTML version. Moreover, some design patterns documents are 'read-only' and may only be opened with a HTML browser. The above conditions influence the decision on what configuration of component, instantiation, anchor and marker objects to use for the representation of the target document. In the Zypher hypermedia framework, such decisions are implemented in the processes that interpret the specifier-quadruples returned by the resolver and turn them into actual component, anchor, instantiation and marker objects. These processes correspond with the #interpretComponentSpec, #interpretAnchorSpec, #interpretComponentAndPSpec, #interpretAnchorAndPSpec messages defined on the path object.

Configuration and Meta Meta objects

The messages #determineResolverFor, #interpretComponentSpec, #interpretAnchorSpec, #interpretComponentAndPSpec, #interpretAnchorAndPSpec defined on the path meta object are part of the navigation layer contract defined on the meta level. As argued in the case of the base level objects ([figure 2]), the mere presence of these messages in the design means that only the static part of the contracts is explicitly available which is not enough to control the execution of the contracts. To attain configuration level tailorability, we must make the dynamic parts of the contracts explicit, which is precisely the role of the hypermedia context object ([figure 5]).

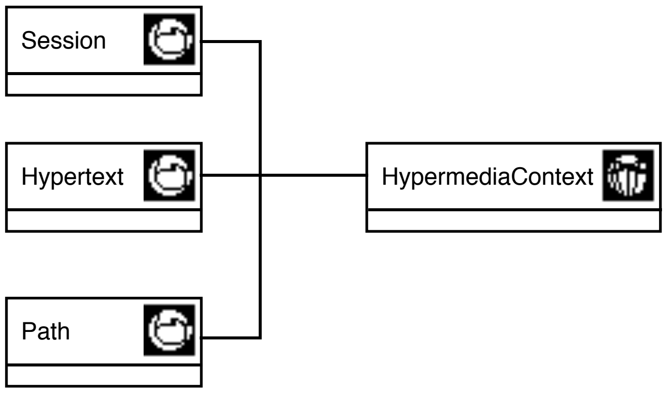


Figure 5: The Hypermedia Context

The implementation of the #determineResolverFor message on the path object must request the HypermediaContext object to return the name of the resolver that must be used (by means of the #determineResolverFor message defined on HypermediaContext). Likewise, the implementation of the #interpretComponentSpec, #interpretAnchorSpec, #interpretComponentAndPSpec, #interpretAnchorAndPSpec must request the HypermediaContext for the name of the classes that should be instantiated (by means of the #determineComponentClassFor, #determineAnchorClassFor, #determineInstantiationClassFor, #determineMarkerClassFor messages).

Just like the meta objects (session, hypertext and path) funnelled all operations dealing with one aspect of the hypermedia system (i.e. presentation, storage and navigation), the HypermediaContext meta meta object funnels all operations controlling the configuration of the hypermedia system. To change the subsequent configuration of base level objects, only one object must be modified.

Note that the parameters passed to, and the results returned from, the messages sent to the HypermediaContext object are always (collections of) strings. This ensures that the system's configuration never depends on particularities of base level objects, which accounts for the 'plug and play' requirement in system level tailorability. Moreover, it allows to implement the configuration messages as look-up tables which are very easy to maintain, even by end users (i.e. compare with the table of 'helper applications' maintained by most World-Wide-Web browsers).

5. Conclusion

One of the ideas that came up during the previous open hypermedia workshop (i.e. [Wiil,Østerbye'94]) was to develop some kind of an 'open hypermedia reference model', similar to what the Dexter model [Halasz,Schwartz'90] did for the generation of monolithic hypermedia systems. The idea behind this paper is to show that, if such an attempt is to succeed, the reference model should embody the notion of system level tailorability (i.e. the incorporation of system services like caching, logging, authority control and integrity control) and configuration level tailorability (i.e. flexible configuration of the system to support run-time extensibility and cross-platform portability). This paper confirms that —although the notions of system and configuration level tailorability may seem quite complex to implement— the technique of a meta object protocol brings it within range of today's software engineering.

The introduction of a meta object protocol has been defined as a process with the following steps. (a) Develop a design specification of an object-oriented framework for an open hypermedia system. Such a specification defines contracts between objects representing the main elements in the design. (b) Define meta objects as an explicit representation of contracts between objects in the base level design. Define a meta object protocol as the protocol specifying how base level objects exchange messages with meta objects. (c) Define a meta meta object as the explicit representation of contracts between objects in the meta level and classes in the base level. Define a meta meta object protocol to establish associations between objects in the base level.

The three levels of tailorability match nicely with the stepwise introduction of the meta object protocol. I.e. step (a) corresponds with domain level tailorability, step (b) accounts for system level tailorability, and step (c) enables configuration level tailorability. What is particularly appealing in the light of an 'Open Hypermedia Reference Model' is the fact that —since steps (b) and (c) are quite independent from the particular design resulting from step (a)— the meta object approach is also applicable to other open hypermedia systems implemented with an object-oriented framework.

6. References

[Beck,Johnson'94] Beck, K. / Johnson, R. "Patterns Generate Architecture"; ECOOP'94 Proceedings, Lecture Notes in Computer Science nr. 821, Springer-Verlag, 1994. Check http://st-www.cs.uiuc.edu /users /patterns /patterns.html for anonymous ftp.

[Berners-LeeEtAl'94] Berners-Lee / Cailliau, R. / Luotonen, A. / Nielsen, H. F. / Secret, A. "The World-Wide Web"; Communications of the ACM - Vol. 37(8) - August '94.

[Blaha,Premerlandi,Rumbaugh'88] Blaha, M. R. / Premerlandi, W. J. / Rumbaugh, J. E. "Relational Database Design Using an Object-Oriented Methodology"; Communications of the ACM - Vol. 31(4) - April '88.

[Conklin'87] Conklin, J. "Hypertext: An Introduction and Survey"; IEEE Computer - Vol. 20 (9) - September 1987.

[Demeyer'96] Demeyer, S. "Zypher: A Hypermedia System Incarnated In a Framework Browser"; Phd. dissertation, forthcoming. Check http://progwww.vub.ac.be/zypher/.

[GammaEtAl'93] Gamma, E. / Helm R. / Johnson R. / Vlissides, J. "Design Patterns: Abstraction and Reuse in Object-Oriented Designs"; ECOOP'93 Proceedings, Lecture Notes in Computer Science nr. 707, Springer-Verlag, 1993. The same people have written the book "Design Patterns"; Addisson-Wesley, 1995.

[Halasz,Schwartz'90] Halasz, F. / Schwartz, M. "The Dexter Hypertext Reference Model"; Proceedings of the 1990 NIST Hypertext Standardisation Workshop (January 16-18, Gaithersburg, MD). Republished in Communications of the ACM - Vol. 37(2) - February '94.

[Johnson'92] Johnson, R. "Documenting Frameworks Using Patterns"; OOPSLA'92 Proceedings, ACM Press, 1992. Check http://st-www.cs.uiuc.edu /users /patterns /patterns.html for anonymous ftp.

[Kiczalis,Rivières,Bobrow'91] Kiczalis, G. / Rivières, J. / Bobrow, D. G. "The Art of the Metaobject Protocol"; MIT Press, 1991.

[Maes'87] Maes, Pattie "Concepts and Experiments in Computational Reflection"; OOPSLA'87 Proceedings, ACM Press, 1987.

[Pree'94] Pree, W. "Design Patterns for Object-Oriented Software Development"; Addisson-Wesley 1994.

[Rao'91] Rao, R. "Implementational Reflection in Silica"; ECOOP'91 Proceedings, Lecture Notes in Computer Science, P. America (Ed.), Springer-Verlag, 1991.

[RumbaugEtAl'91] Rumbaugh, J. Blaha, M. / Premerlandi, W. / Eddy, F. / Lorenson, W. "Object-Oriented Modeling and Design"; Prentice Hall, 1991.

[Steyaert'94] Steyaert, P. "Open Design of Object-Oriented Languages"; Phd. dissertation, Vrije Universiteit Brussel, 1994. Check http://progwww.vub.ac.be/prog/papers/paperquery.

[Wiil,Østerbye'94] Wiil, U. K. / Østerbye, K (editors) "Proceedings of the ECHT'94 Workshop on Open Hypermedia Systems"; Technical report R-94-2038 / Institute for Electronic Systems Department of Mathematics and Computer Science - Fredrik bajers Vej 7 - DK 9220 Aalborg - Denmark. Check ftp://ftp.iesd.auc.dk/pub/packages/hypertext/ECHT94-workshop/.

[Zellweger'89] Zellweger, P. T. "Scripted Documents: A Hypermedia Path Mechanism"; Hypertext'89 Proceedings, ACM Press, 1992.