# Indirection Pattern
for Data Modeling

Birthe Böhm, Norbert Gewald
Corporate Technology, Siemens AG
Günther-Scharowsky-Str. 1, 91050 Erlangen, Germany
{birthe.boehm, norbert.gewald}@siemens.com

Gerold Herold
Medical Solutions, Siemens AG
Hartmannstr. 16, 91050 Erlangen, Germany
gerold.herold@siemens.com

Dieter Wißmann
Department of Electrical Engineering and Computer Sciences
University of Applied Sciences Coburg
Friedrich-Streib-Str. 2, 96450 Coburg, Germany
wissmann@fh-coburg.de

Abstract

Current architectural concepts in general demand the decomposition of a system into independent modules with well-defined interfaces. These concepts lead to a decoupling of data from the rest of the system by hiding the data access details behind an abstract interface. This pattern describes how the data access itself to exchangeable data resources can be designed in such a way that the data resources can be substituted or used alternatively at runtime of a system. For example, this pattern can be applied either to embed multilingual texts in language-independent data or into software applications.

Intent

The indirection pattern is used to decouple a software application or some parent data from logically embedded data resources, whilst at the same time maintaining the relation between them. The data resources can be exchanged without having to alter the original application or parent data by applying an indirection concept.

In the following the pattern is described from the application's point of view, but the pure modeling of the underlying data resources of this pattern can be done also independently from the application as it is explained in the implementation section.

Example: An application has some data which contain also texts in English. Since these data must also be understood in Germany we want to provide German translations for all English texts. But instead of simply exchanging all text elements in the data into their German translations, we want to provide the option to switch between the two languages. That is, it should be possible to use the German texts but, later on, include their English versions without having to adapt the data. Can these text elements be stored in such a way that it is possible to switch between languages in the data? Can additional languages be added without altering the project data? How should the resources be structured and used by the application to decouple them as far as possible?

Context

Use this pattern when the relation between an application or some parent data and its resources should be stored both explicitly and flexibly at the same time, so that it is possible to exchange these resources quickly and without adapting the application itself.

Problem

Experience from long living software systems shows that data and their underlying data models are an important aspect of architecture and systems design. This is because data usually lives significantly longer than the software system the data was designed for, and because the data is processed by many different systems. Thus, a consistent definition and usage of data resources is required in order to retain control over this data over time. A good approach is to separate this data from the accessing applications. Separation is used in many areas, e.g. for supporting different languages or device-specific symbol libraries for display on different devices like PC, PDA or cell phone. Often the separation is achieved by referencing a resource file and exchanging it when the resource should be altered or substituted. The elements in the resource file are found by identifier inside the file.

With this concept, we have a decoupling of the resources from the application, but there is an issue as the application is still bound to the resources.

If we want to store different variations of the same resource and choose the correct resource on the basis of a configuration context we face some problems. E.g. an application should be presented to the user in a different language depending on the language of the operating environment. If the relation to the variations of the resource was given in the application, there would be multiple references to the variations of the resource. Therefore we would have to alter the application always when we added a new language later on to establish the references to the new language resource as well.

What we really want to achieve is decoupling the applications from their resources – based on a concept that enables an abstraction of the software implementation details. In order to support the decoupling while still keeping up the relations, we introduce another concept as it is described below.

A successful solution to this problem will resolve the following forces:

- Data resources need to be kept independent from the processing software applications. Reasons might be the possibility to allow these resources to be easily swapped for alternate resources based on the context or because of these resources should live longer than their environment.

- Dependencies between resources and applications shall be avoided.
- Data resources that are provided from different sources and at a different point in time can be attached to an already existing application without any adaptations.
- If the relation to the resources is given in the application then the application should not be altered in case of resource changes. Such changes may arise because additional resources are added or a resource is moved or renamed.

Solution

Introduce an indirection table between the application and the separated data resources. This indirection table maps specific values of a configuration option to a reference of one of the separated resources. That is, the indirection table maintains references to all resources that are addressed via the indirection table. These resources are structured identically so that they map resource keys to their concrete values.

In any application dealing with this concept, implement an algorithm that looks up the adequate resource by means of a configuration option in the indirection table. For each defined configuration option value, a reference to a resource is retrieved. This resource shall be used by the algorithm to look up the required value for a given resource key. Figure 1 shows the concept described so far.

The configuration option is either set globally or it might be also provided as an input parameter to the indirection table. Changing the configuration option means retrieving a reference to another resource and accessing this resource with the given resource key instead. That means that a different value is retrieved based on a different configuration option but with identical resource key.

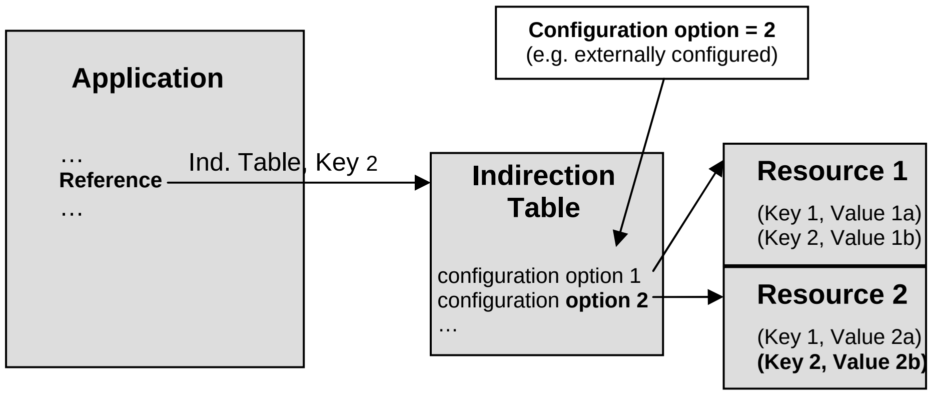

Figure 1 : The concept of the indirection pattern

Figure 1 shows a sketch of the concept. The application references the indirection table and provides a key ("Key 2"). When resolving this reference, the configuration option (configuration option 2) is used to look up the corresponding resource in the indirection table (Resource 2). In this resource, the key is found that contains the desired value ("Value 2b"). This value is returned to the application and is used as value of the given reference.

A reference in the application consists of two parts, a part which gives the indirection table name and, if required, a key to locate the resource in the resource file. If the resource has no sub-element but is used as a whole the key can be omitted.

The indirection table contains references to the resources. That is, each configuration option refers to exactly one resource. If the resources themselves are not used as a whole then they should be sub structured. Sub structured resources contain key-value-pairs as it is shown in the above figure.

Consequences

Benefits:

- The pattern decouples the application from its resources. This means the resources can be exchanged, altered, or updated more flexibly.
- There are no direct dependencies between resources and their applications that must be maintained. Due to the generic access to the data resources and the indirection step there is a well-defined separation of the implementation of the application and the resource data.
- Such decoupled resources can be maintained by different sources without additional effort. This makes it possible that the resources are supplied by different sources and at varying point in times. E.g. non-technical staff as translators or a group of different stakeholders provide each an own resource. These resources can be attached easily without having to adapt the overall application.
- The application that maintains relations to the independent data resources do not need to be adapted in case of changes to the data resources, in general. In the worst case, only the handling of the indirection table and the setting of the outside configuration option has to be adapted – the rest of the application remains untouched.

Liabilities:

- Dynamic aspects are not covered – the intelligent binding of possibly already existing data resources shall be addressed instead since the structuring of data is an additional challenge that is often neglected.
- The indirection table has to be maintained with up-to-date references to the resources and the resources must be consistent to each other in terms of the keys that can be retrieved inside.
- Finding the correct resource in the indirection table might be an overhead.
- There are some rules concerning the structure of the resources and the access rules to these resources which have to be defined once and obeyed by all participating resources and applications. I.e. on the one hand, each of the applications that access the resource data needs to know how the resource access has to be conducted and how to access the resources. On the other hand, it is necessary that all resources have to be structured in the same way to make sure that they are interchangeable.

Implementation

Application-centric View

The diagram in Figure 2 shows the concept as described in the section solution as an UML diagram. It shows an implementation of the concept sketch given in Figure 1: In order to access resource elements, the application (*Application*) uses a resource reference (*ResourceReference*) to identify the required indirection table (*IndirectionTable*) according to the *IndirectionTableName* element of the resource reference. This table contains an arbitrary number of alternatives, our so-called configuration options (*Alternative*). According to a preset configuration option, one of these alternatives is chosen. This alternative again refers to a resource that finally contains the resource elements which are in fact a key (*Key*) and value (*Value*) pair. The required resource element is chosen by looking up the key given as member *Key* in the initially passed resource reference.

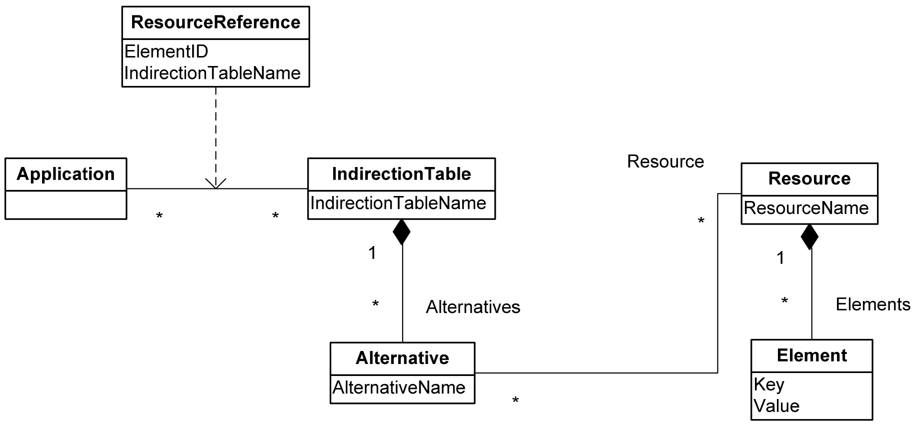

Figure 2 : Schematic view of the indirection concept

In addition to the implementation of the algorithms, the design of the data structures also plays an important role. This is because we want to achieve independency from the specific implementation of the applications. Therefore all static and permanent data used by the applications at runtime are modeled as application-independent data structures. Such data structures are provided for the instantiations of the indirection table, the alternatives, resources and elements.

Data-centric View

A more data-centric view is useful for data modeling purposes. In this case the structuring of the data is in the focus but all access-related algorithms and therefore application runtime aspects are left out. The algorithmic aspects described above are then only stated as rules how to process the given data while the definition of the data structures is emphasized.

Example: This data-centric view is explained with the following example. The structure of the indirection table in a graphical XML Schema representation can be seen in Figure 3. The indirection table itself may possess a description and a list of references to the available resources together with an element that gives its type. This type element is used to look up the reference to the appropriate resource. After having retrieved this resource, it is possible to actually get the resource entry we were looking for.

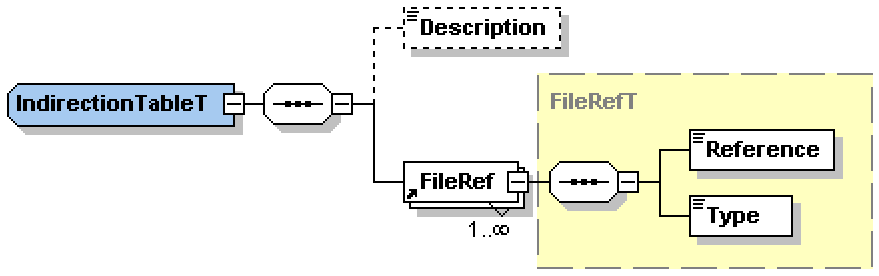

Figure 3 : Graphical XML Schema example

In Figure 4 the indirection concept is used to decouple text resources from a document. The example can easily be adapted for other kinds of resources.

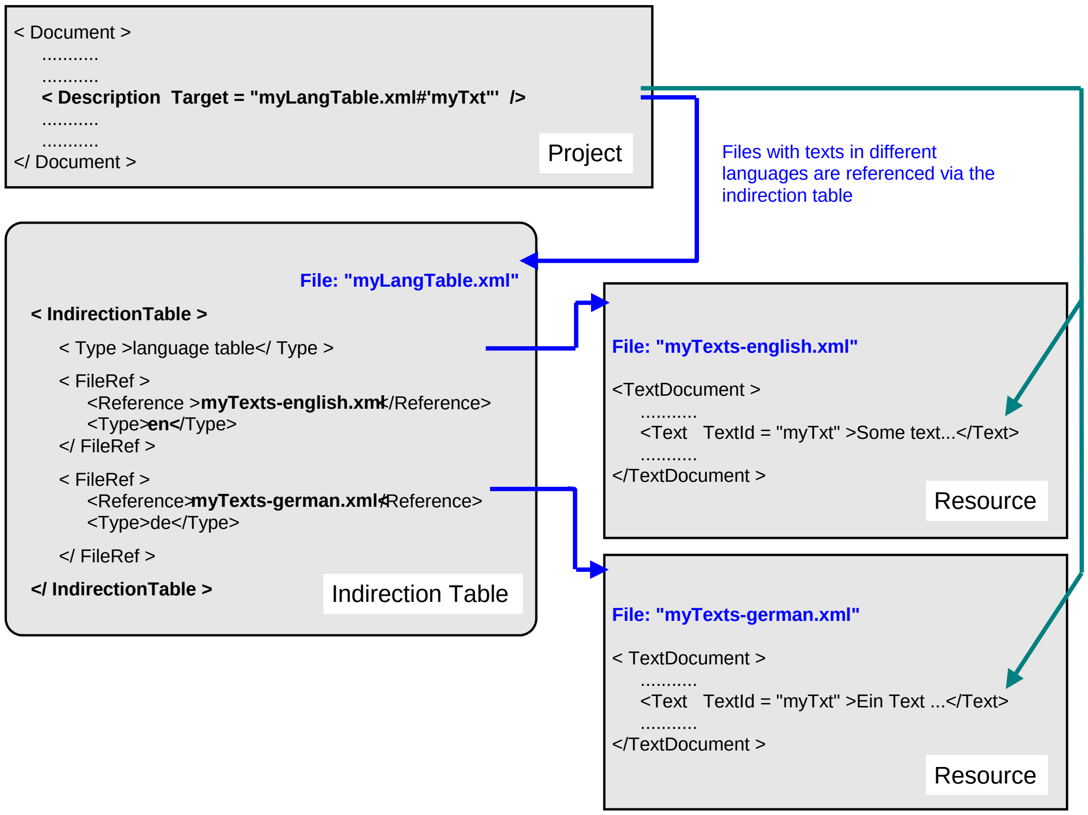
< Document >
....
....
< Description Target = "myLangTable.xml#'myTxt"' />
....
....
</ Document >
Project
Files with texts in different languages are referenced via the indirection table
File: "myLangTable.xml"
< IndirectionTable >
< Type >language table</ Type >
< FileRef >
<Reference >myTexts-english.xml</Reference>
<Type>en</Type>
</ FileRef >
< FileRef >
<Reference>myTexts-german.xml</Reference>
<Type>de</Type>
</ FileRef >
</ IndirectionTable >
Indirection Table
File: "myTexts-english.xml"
<TextDocument >
....
<Text TextId = "myTxt" >Some text...</Text>
....
</TextDocument >
Resource
File: "myTexts-german.xml"
< TextDocument >
....
<Text TextId = "myTxt" >Ein Text ...</Text>
....
</TextDocument >
Resource

Figure 4 : Text resource example

The texts of the parent data are stored in a library. The library is available in both English and German.

The parent data contains references to the texts. Each reference is constructed of two parts separated by '#', in our example the reference equals to "myLangTable.xml#myTxt". The first part specifies the indirection table "myLangTable.xml", the second the resource identity (ID) which actually points to the text entry inside the resource file "myTxt".

The indirection table stores the locations of the variations of the resource. The FileRef element contains the reference to one resource and the type contains the language of the resource. In this example the options "en" for the English resource and "de" for the German resource are possible. An outside configuration option must be available to choose the right language, i.e. must be set to one of the above values. Alternatively the language could be passed as an input to the application retrieving the indirection table as well. Let us assume that in this case the configuration option is set to "en" and hence the resource "myTexts-english.xml" is chosen in the indirection table.

Once the right resource is determined, the text ID is used to get the entry out of this resource. All texts inside the resource have a *TextId* attribute which implements this text ID and uniquely identifies this text. Since our *TextId* equals "myTxt" the text "Some text…" is retrieved in the resource myTxts-english.xml".

If the configuration option was set to "de" then the text "Ein Text…" would have been retrieved by using the same *TextId*.

If additional language resources are added, a reference to them needs only to be added to the indirection table. The appropriate language can be selected via external configuration and no changes to the parent data are necessary.

Resource standardization

Although this pattern describes the required dynamic behavior and the necessary data structures, an implementation of this pattern can differ from another implementation in some details, e.g. regarding data types or element names. Therefore if this pattern is employed for a system in which a number of applications are using this pattern it should be carefully considered that a common understanding is agreed on the data structures in addition to the access algorithms, i.e. standardization of the data structures is necessary in such an environment.

Application of indirection tables

The pattern can also be applied at a later stage of development. For example, in the beginning there may be only one resource. Later on, additional variations may occur. Then the indirection table can be introduced. The introduction of an indirection table and external resources will generate some additional effort but in the long term this effort will be rewarded with less maintenance costs. In a broader sense, even if there are no plans to use variations on a resource the indirection concept can be used as a form of future-proofing.

Moving the data to a different environment

To be able to move the whole solution (including the indirection and the resources) to a different environment, e.g. for back-up or to a different platform, the parent data and the indirection table should use relative references as opposed to references that depend on e.g. the concrete location in a file system. Otherwise the references in the parent data and the indirection table have to be altered or the lookup mechanism has to be adapted to deal with the different environment.

Variants

Replacing resources

In case a resource is sub-structured, there will be a number of elements within the resource. If this resource is to be replaced by another resource, then the replacement resource must contain all the elements that were in the original resource. These elements should have the same *Key* as the original elements and should differ only in content.

Example: A resource that maps keys to English texts could be replaced by a resource that maps keys to Italian texts. Since an application or parent data accesses the English texts by using the keys as identifier, the Italian text resource must contain the same keys but the Italian texts instead. It should not omit some of the keys used in the English text resource or use other keys or another concept for retrieving the texts. Only then, when a replacement resource is selected, is it always possible to find the correct replacement element no matter which resource has been selected in the configuration options.

Default Resource

So far, we have assumed in this pattern that all resources contain for each of the defined keys a resource element. That is, all keys must be available in all resources. This pattern can easily be extended to another use case. There may be a default resource and only some entries should be replaced for specific applications while the original resource shall be kept. So a new resource would only offer the changed entries and for the missing entries the default resource is used. To do this, the default resource has to be marked explicitly and the retrieval algorithm has to look up missing keys in the default resource if they cannot be found in the originally selected resource.

Example: Assume that there is a German and an Austrian language resource. Most entries in these resources will be the same because German and Austrian are closely related languages but some entries will differ. As a consequence, a German language resource could be created that covers all entries while the Austrian language resource only covers those entries that really differ. In case the Austrian language option is set, the retrieval algorithm would access this resource at first. If the required key is found in it then it would return the corresponding element otherwise the key would be looked up in the German language resource.

The clear advantage of this approach is that each text element only needs to be entered and maintained once.

Additional indirection steps

The concept of indirection can be applied repeatedly. An entry in an indirection table can point not only to a real resource, but to another indirection table. This may be useful if more than one selection criterion exists: For example, if a resource is not only selected by its language but also by the kind of output device. Then a first indirection table might select another indirection table by the current language. The next indirection table gets the required resource by looking for the required output device.

See Also

The Lookup pattern by Kircher and Jain (2004) addresses the dynamic aspects of resolving names into service references and therefore also employs an indirection step. Although these two patterns follow the same overall principle of indirection, the Lookup pattern focuses on the algorithm implemented by the participating components and not on structuring of data. In the context of the Indirection pattern, the Lookup pattern could be used to implement the access to the resources. Vice versa, the Indirection pattern could be used as a guideline for the implementation of the Lookup service itself (that exactly looks up the reference to a service that belongs to a given name).

Known Uses

In some development environments this concept is implemented for language resources, e.g. Microsoft Visual Studio. Different language resources can be created that contain key-value-elements. When an application accesses such a resource element by its key at runtime, then at first the language resource is chosen according to the active system language. Then the value of the resource element is looked up inside of this resource using the key.

The indirection pattern was also implemented for a standard data format developed by Siemens Building Technologies that describes data interchanged between engineering tools for building automation. This data contains texts in natural languages among language-independent data. For all texts in natural languages the indirection pattern is used to enable the coexistence of different languages.

Acknowledgment

We would like to thank Michael Kircher for his early and valuable input for writing patterns and Frank Buschmann for his excellent coaching und the fruitful discussions during the shepherding for the EuroPLoP 2007 conference.

Literature

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal: *Pattern Oriented Software Architecture Volume 1: A System of Patterns*, Wiley, 1996

E. Gamma, R. Helm, R Johnson and J Vlissides: *Design Patterns: Elements of Reusable Design*, Addison-Wesley, 1995

M. Fowler: *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1997

M. Fowler et al: *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003

M. Kircher, Prashant Jain: *Pattern Oriented Software Architecture Volume 3: Patterns for Resource Management*, Wiley, 2004

D. Schmidt, M. Stal, H. Rohnert, F Buschmann: *Pattern Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*, Wiley, 2000