Architecting Value

IT SERVICES | BPO | SOLUTIONS

Data Management Requirements
A Use Case Pattern Language

White Paper produced by
MphasiS Corporation

March 2005
Version 1.1

© MphasiS. All rights reserved

MphasiS
an HP company

Document Contact Information

Name	Bert Hooyman, Senior Architect MphasiS UK Ltd. Southbank House Black Prince Road London SE1 7SJ, United Kingdom
Email	bert.hooyman@mphasis.com
Business Phone	+31 618 780 559
Business Fax	+31 302 967 801

Table 1 Contact Details

Note From The Author

The use case pattern described in this white paper is the result of exposure to numerous recurrences of the same data management requirements. We believe the pattern language covers all the essentials of data management. Having said that, we find that reality is always more complex than we are led to believe. Many of these 'reality checks' have found their way in this white paper by way of "points to consider", included in the use case descriptions in chapter 3. We invite all readers to actively contribute additional reality checks and contradictory findings, with the purpose of further enhancing the quality of this work.

Acknowledgements

I have had much help and valuable feedback from the following MphasiS BAG Practice members: Madhavi Soman, Jai Sheth, Tarang Baxi and Pooja Badve. I thank them all for helping put this white paper together.

Changes in this version

For version 1.1 of the white paper, the original use case title "Browse Whatevers" has been replaced by "Filter Whatevers". As it turned out, the verb *browsing* caused more confusion than expected, and the alternative *filtering* was suggested as a more intuitive description. As a result, some changes have been made to section 3.3.2 of this white paper.

© MphasiS Corporation

Contents

Document Contact Information i
Note From The Author i
Acknowledgements i
Changes in this Version i
1 Executive Summary 1
Audience & Objectives 1
Objectives 1
Prerequisites 1
Further Reading 2
2 Business Requirements 3
2.1 A Business Example 3
3 Solution Outline 8
3.1 Creation & Modification of Whatevers 8
3.2 Viewing Whatevers 11
3.3 Searching for Whatevers 13
3.4 Extension Points 14
3.5 Pattern Variations 15
4 MphasiS Case Study - Web-Enabled Order Visibility 18
4.1 Business Objectives 18
4.2 Project Overview 18
4.3 MphasiS Approach 19
4.4 Application of the Pattern Language 20
4.5 Bottom Line 21
5 Conclusions 23
Appendix A - References 26

List of Figures

Figure 1 Early Analytical Class Diagram for Library Project 4
Figure 2 Improved Analytical Class Diagram 4
Figure 3 ER Diagram for Library 5
Figure 4 Database Model for Library 6
Figure 5 Catalog Browser at BestBuy 13
Figure 6 Use Case Diagram for Manage a Whatever 23

© MphasiS Corporation

1 Executive Summary

This MphasiS white paper addresses the management of data in web-enabled business applications. The scope of this research is all those parts of a web-enabled application that are light on behavior and heavy on data. This includes, but is not limited to, screen population, data validation and entitlement management. In many cases, the primary business function of an application is indeed little more than data management. An example of this is the web-enabling of corporate ERP solutions, where the business process automation, the workflows and the resource planning activities are centralized in the ERP system but data entry and query is pushed out to the corporate intranet.

In this white paper, we present a requirements analysis and derive generally applicable use cases. These use case together describe a pattern which we refer to as "Manage a Whatever". Once the pattern is understood, it is easy to identify during real-life business requirements analysis activities. The pattern helps exposing the peculiarities of each type of data and assists in properly scoping the work needed for data management. Flexibility is provided through pattern variations, which are simplifications or extensions of the baseline pattern.

Audience & Objectives

This white paper is written for MphasiS clients. It describes a pattern language for capturing the requirements of data management tasks. Using the pattern language, MphasiS business analysts can quickly identify client requirements w.r.t. data management, identify omissions, duplications and potential simplifications. With the use case analysis in hand, follow-on activities such as effort estimation, usability engineering, test script generation and software design can build on the formal description of the use case pattern language.

Objectives

The goal of this white paper is to introduce a formal analysis method for the recurring requirements around data management. The method is based on a pattern language that helps defining the precise business requirements. Using the pattern language, business analysts should be able to quickly identify client requirements w.r.t. data management, identify omissions, duplications and potential simplifications. In the concluding chapter of this report, I present an example of a *requirements matrix* that condenses the business requirements for data management in a simple tabular format.

Prerequisites

A general awareness of use case analysis is required to follow the analysis in chapter 3.

© MphasiS Corporation

In section 2.1, I present an example scenario. The example includes some UML class diagrams, an Entity-Relationship model and a database diagram. Don't let these diagrams scare you – it is OK if you miss out on some of the finer details there.

Further Reading

In Appendix A - References, I include some valuable literature references.

© MphasiS Corporation
MPHASIS
an HP company

2 Business Requirements

Requirements for data management frequently show up as 'screen population' requirements in complex data-entry environments. To support the data entry task at hand, reducing data entry errors and ensuring valid data entry, many user interface designs focus on selecting values from enumerations, instead of letting users type in codes, names or numbers.

This approach is further encouraged by the desire to derive management information from the data that was entered. For the purpose of MIS, it is essential to work with enumerated lists. Reports that show "Downloads by country", or "Purchases by segment" require that the lists of countries and segments, respectively, are under control of the application used to enter the information and not under control of the person entering the data.

Implicit in the above requirements analysis is the assumption that the system under study will provide a means to manage the list of countries and the list of segments. Some of these lists are of a static nature and really do not require any administrative support at all. A list of countries is a good example of this.

In the case of consumer segmentation, the list of segments is usually under control of a marketing organization. Changes to the segmentation may come with the season. Administrative access to the list of segments becomes part of the functional requirements of the system under study. Now, the functional requirement that 'visitors can select a profile from a given list' suddenly implies 'marketing staff can add, modify and remove profiles from the segment list'. During the requirements analysis phase, this explodes into one or more additional use cases that cover the administrative access to the segments list.

Look for data management requirements wherever the system under study identifies enumerations. Screen population requirements and data validation requirements are two natural places to find enumerated values. More *master data* may be present in the access control part of the system under study. Roles, entitlements, resources, users and groups can be qualified as master data in many cases. The source of master data is not really important at this point - it can either be stored in a database designed to support the system under study, or in an existing (legacy) database, a registry, a directory or made available as a Web Service. The requirements for data management may exist for any of those.

Beyond master data as identified above, many web-based applications are little more than data management front-ends to (legacy) systems. In these cases, all the primary business objects are further candidates for the Data Management pattern language.

2.1 A Business Example

To illustrate the intended scope of a use case pattern language, this section describes a typical web-enabled business application in terms of its data management requirements. The subject area of this example is a library of books.

© MphasiS Corporation

The library holds books (publications) and for these books we need publishers, keywords, authors and categories. The system is set up to support a 'virtual' library, run by a team of employees in an organization. The participants all register the books they own and provide them to the others to read.

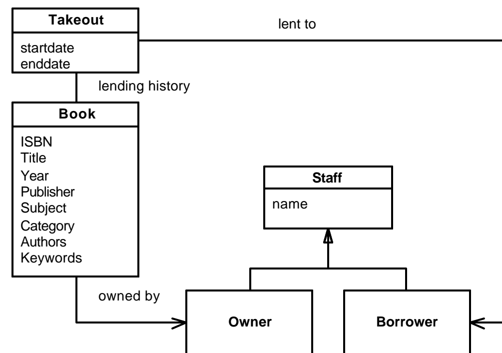

Figure 1 Early Analytical Class Diagram for Library Project

The first class diagram, shown in Figure 1, models Books, Takeouts, Owners and Borrowers. It is recognized that both Owners and Borrowers are specializations of employees (Staff). Every Takeout registers the lending of one particular book by a borrower on a date. The return date is also captured as part of the Takeout. It doesn't take long to realize that there may be multiple copies of a book, and each copy may be lent out separately. As a consequence, each copy has an owner. So the Book class should be replaced by a Publication and one or more Copies. At the same time, Owner and Borrower can be removed from the class diagram because there are no apparent methods or attributes that are unique for either of the two.

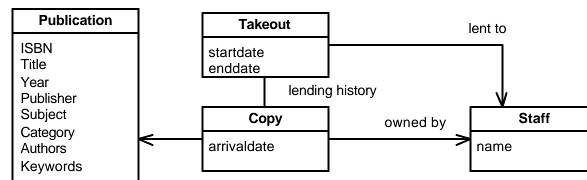

Figure 2 Improved Analytical Class Diagram

From the requirements perspective, the improved class diagram in Figure 2 is sufficient. The use cases for the library evolve around taking out and returning copies of publications, adding new publications to the library and tracing late returns

During the technical design, several cases of implied data management are found in the class diagram. The class diagram is transformed into an ER diagram (Figure 3) that makes the data items explicit. The business users agree with this because they can see the benefit of controlling keywords, publishers, authors and catego-

© MphasiS Corporation

ries separate from the publications. They are not very happy with all the new use cases, though…

In the end, Publication, Keyword, Author, Category, Subject, Publisher and Copy are identified as target data entities. Takeout is the primary domain object for the library application. Staff is recognized as master data without data management requirements, as the data already exists in the software infrastructure.

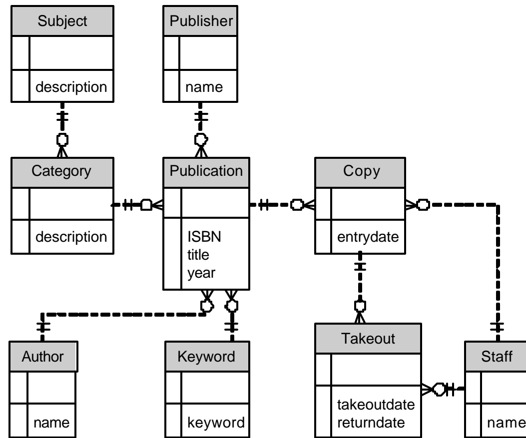

Figure 3 ER Diagram for Library

During detailed technical analysis two n:m relationships are identified that require extra effort in the data model. The final data model, shown in Figure 4, includes 11 tables, against 4 classes in the class diagram.

As can be seen in the data model, Keyword has an n:m relationship to Publication (meaning a keyword can apply to multiple publications, and multiple keywords can apply to the same publication). Likewise for Author.

Category and Publisher both have a 1:n relationship with Publication (each Publication is published by at most one Publisher).

Copy has an n:1 relationship with Publication: there can be multiple Copies of a single Publication. This is a true master/child relationship, comparable to orders and order line items. Ownership of copies uses the Copy to Staff relationship which is 1:n (each Copy has a single Owner) Takeout of copies from the library is registered in Takeout. Each instance of Takeout links exactly one Copy to exactly one Staff member.

The following entities are prime candidates for Manage a Whatever:

- *Subject*, *Publisher*, *Keyword* and *Author* are all data items of the simplest kind – their attributes are all free-format text strings. There are no further enumerated value lists associated with these four entities.
  Of these, Subject is an entity that is not expected to change very often, this is a super-classifier for Publication Categories and as such it is most likely that once this has been defined, there will be no need for any updates. So this entity may

© MphasiS Corporation
MphasiS an HP company

eventually not require any management at all (delegated to the database administrator).

- *Category* is a data item with a text attribute (the name of the category) and an association with Subject (which is a 1:n relationship).

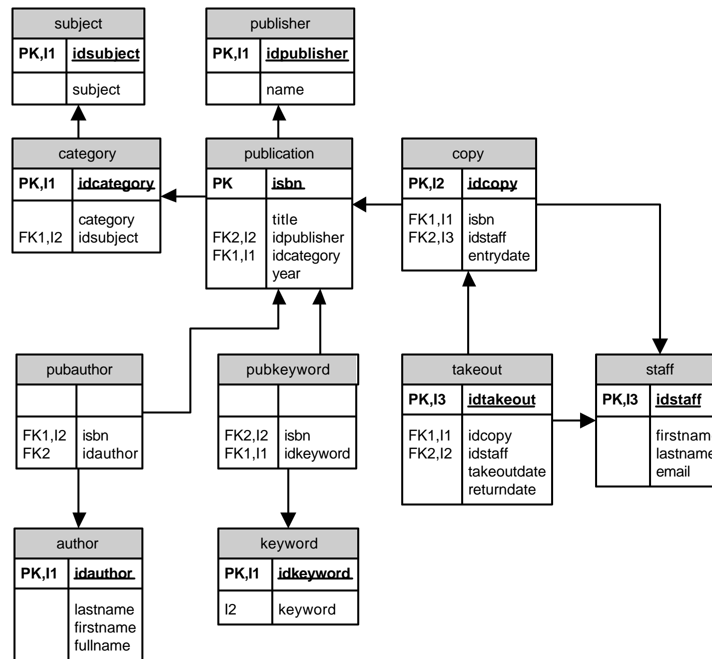

Figure 4 Database Model for Library

- *Publication* is the richest entity of the system, which is clearly shown in the diagram. Publication holds three attributes plus two references to other entities and is referenced from another three entities. This is definitely the most interesting entity to cover under "Manage a Whatever", and this will be used as an example throughout this document. Note that of all the attributes, only the ISBN number, the publication title and the year of publication require data entry. All other attributes are potentially available from pre-populated selection lists.
- *Copy* is the detail entity in a master/detail relationship with Publication. The creation of new instances of Copy should ideally be tied in with the management of Publication instances.
  When managed in isolation, the reference to Publication will either be a text-entry element, which requires subsequent validation, or a selection from a list of existing publications which may be cumbersome because it's a long list, potentially. For these reasons, it seems more appropriate to use "Manage a Publication" to identify a publication of choice and add a new instance of Copy in

© MphasiS Corporation
MPHASIS an HP company

that context. This would be a combination of “Find a Publication” and “Update a Publication” [1].

- *Takeout* is the core Entity for running the library. Every takeout of a book (an instance of Copy) must be registered as an instance of Takeout. The database model holds date of takeout as well as date of return. Null values for date of return are used to model books currently out on loan. This entity may be considered in scope for “Manage a Whatever” but, depending on the business requirements, a more elaborate set of Use Cases could be needed. Note that the attributes of Takeout are all coming either from enumerated lists (lender, copy) or through automatic entry (takeout date, return date).
- *Staff* is referenced in Copy and in Takeout but not managed within the scope of the system under study (the Staff entities are shared across many applications in this organization). So although the system must provide a list of staff members for creating publication copies and for the lending administration, it is not the responsibility of this system to create or delete users or otherwise modify user entities. In section 3.5.4 of this report, we refer to such entities as a read-only domain.
- *Pubauthor* and *Pubkeyword* are merge tables that do not play a role in the Entity Relationship model. These data items are only needed to implement an n:m relationship. As such, instances of these get created when authors (or keywords) are linked to publications.

[1] These are two of the use cases that are part of the pattern language. They are covered in sections 3.3.1 and 3.1.3, respectively.

© MphasiS Corporation

MphasiS an HP company

3 Solution Outline

In chapter 2, we have seen how a preliminary class diagram for a lending library leads to a functional requirement for data management. Once the need is established, there is actually a *pattern of use cases* that may occur. The pattern encompasses the full lifecycle of the data and includes creation, query, update and delete (CRUD) as well as various listings, sorting, printing, exporting and more.

Interestingly, the pattern applies almost universally to all types of data, and indeed in many cases also to the primary *Domain Objects* of the system under study. In terms of object-oriented design, the pattern applies to all those objects that are light on behavior and heavy on data. Given the general applicability of the pattern, we refer to it as *Manage a Whatever*. In his excellent book "Writing Effective Use Cases" (see reference [1]), Alistair Cockburn identifies these same use cases as Manage a Frizzle. He further discusses the notion of *parameterized* use cases, a subject we also consider to cover in a subsequent white paper.

In the following sections, we will identify all the separate use cases that together make up the Manage a Whatever pattern. In a large-scale requirement analysis, these use cases are sometimes pulled together and identified simply as "Manage a Whatever". We believe that it makes sense to do so given the overall complexity of the system under study, but the simplifications fails to communicate at least three aspects of the requirements:

- Which requirements are 'must have' versus 'should have' or 'could have'
- The effort required for design and implementation
- Who has access to which use cases (security constraints)

Therefore, we believe it makes more sense to qualify the requirements and refer to the individual use cases presented below.

Note that when a use case document is created for such a large and complex system, it would be perfectly acceptable to include the use cases as given here only once, identifying them as generic solutions.

3.1 Creation & Modification of Whatevers

Use Cases in this section describe the core goals of data management use cases, oftentimes referred to as *CRUD* operations (for Create, Retrieve, Update and Delete).

3.1.1 Create a New Whatever

Use Case Brief

In the context of data management, the ability to create new instances of a data entity are an important user goal. This use case addresses that goal.

© MphasiS Corporation
MphasiS an HP company

Success Guarantees

A new instance of Whatever is persisted and made available to other application users.

Points to Consider

Depending on the business requirements, the system may generate a unique entity identifier before the data entry form is displayed. One example of this behavior is in trouble ticket issuing, where the ticket ID is already shown on the data entry page because that information needs to be exchanged with the client.

Although it may sound contradictory in the context of data management, in some situations the source of the data is not in scope of the application and no interfaces are available to modify data or create new instances. We refer to this as a *read-only domain*. Examples of this scenario are Directory Services, Repositories and external systems that are Web-Service enabled but only for viewing.

Data elements with a master-detail relationship require special attention. Because detail entities are always created in the context of their master, it makes sense to integrate *Create a Detail* with Create a new Master or Update a Master. This makes the identification of the master entity implicit which is a good thing. Depending on the complexity of Create a Detail the task may require a separate screen or the detail entity may be created on the same page where the data is present.

3.1.2 Copy a Whatever

Use Case Brief

In the context of data management, the ability to create new instances of a data entity are an important user goal. One way to satisfy that goal is to (partially) copy an existing instance. This goal typically only exists for entities with many attributes or extended attribute collections.

Success Guarantees

A new instance of Whatever is persisted and made available to other application users.

Points to Consider

Not all attributes would normally be copied – those that are unique to an instance usually are not. The exact set of attributes to copy would be determined by the business logic.

This Use Case is only valid in context where a "Current Whatever" is identified.

Although it may sound contradictory in the context of data management, in some situations the source of the data is not in scope of the application and no interfaces are available to modify data or create new instances (read-only domain). Examples of this scenario are Directory Services, Repositories and external systems that are Web-Service enabled but only for viewing.

When the data domain is very simple and only a few attributes are held for each instance, Copy a Whatever does not apply.

© MphasiS Corporation

The data elements for this use case are totally dependent on the data model of 'Whatever'. At the very least, the data elements include all *mandatory* attributes of Whatever. Normally, this use case would allow entry of *all* attributes of Whatever. When Whatever is the master part of a master/detail relationship, creation of detail records would also be acceptable within the context of this use case.

3.1.3 Update a Whatever

Use Case Brief

Application data items may require modifications. Users access the data repository to update individual instances of data entities.

Success Guarantees

The updated instance of Whatever is persisted and made available to other application users.

Points to Consider

Some attributes may not be eligible for updates (primary keys are an example). Precisely which attributes should be updateable is determined by the business logic.

In some situations, business entities are never updated. This is referred to as a *final domain*. An example of this is a master list of currencies. New currency definitions may be defined, but existing currencies are typically never updated.

This Use Case is only valid in context where a "Current Whatever" is identified.

Although it may sound contradictory in the context of data management, in some situations the source of the data is not in scope of the application and no interfaces are available to modify data or create new instances (read-only domain). Examples of this scenario are Directory Services, Repositories and external systems that are Web-Service enabled but only for viewing.

The data elements for this use case are totally dependent on the data model of 'Whatever'. Normally, this use case would allow modification of *all* attributes of Whatever, perhaps with the exception of any immutable keys. When Whatever is the master part of a master/detail relationship, changes to detail records would also be acceptable within the context of this use case.

3.1.4 Delete a Whatever

Use Case Brief

Data items may become obsolete and serve no further purpose in an application. Users access the data repository to remove individual instances of data entities.

Success Guarantees

The selected instance is removed from persistent storage.

© MphasiS Corporation

Points to Consider

In some situations, business entities are never deleted. An example of this is a master list of currencies. New currencies may be defined, but existing currencies are typically never deleted. This is referred to in section 3.5.2 as a *final domain*.

For a delete, the only data element that is really needed is the identifying key of the instance. When a single instance is in context, the key of that instance is the target of the delete.

3.2 Viewing Whatevers

Generally speaking, data instances are viewed in one of two formats: either in a list format or in a detail format. This section describes the two fundamental viewing use cases along with a few supporting use cases.

3.2.1 List Whatevers

Use Case Brief

As part of data management, List Whatevers plays a pivotal role. List Whatevers is almost always a globally visible command, allowing users to jump from anywhere within Manage a Whatever to the List Whatevers page. Also, List Whatevers is invoked as a result of "Find a Whatever" and "Filter Whatevers". Finally, List Whatevers is the starting point for several other use cases, including "View a Whatever".

Success Guarantees

The instance summaries are displayed.

Points to Consider

A web page may display a combination of list and detail views, showing one of the entities from the list in more detail. Master-detail entities usually show the master data along with the detail data on one page as well.

List Whatevers almost invariably needs a paging mechanism, allowing users to view list data on a page-by-page basis. The primary driver for this requirement is the anticipated instance count of the domain. Paging should always include "back to start" and "next page", and usually also includes "previous page" and "to end". Many more elaborate paging schemes are known.

3.2.2 Sort a List of Whatevers

Use Case Brief

As an extension of presenting a list of Whatevers, it is oftentimes useful to sort the list in a particular order, possibly as a prelude to "View a Whatever". Search or filter the domain first, then sort the result set and select the first instance. Use "Next Whatever" to page through the result set in the desired order.

Success Guarantees

The sorted instance summaries are displayed.

© MphasiS Corporation

Points to Consider

For each attribute to sort on, the collating sequence must be known.

This use case is an extension of "List Whatevers". It may apply to all entities with more than just a few instances, although some entities simply have no attributes to sort on.

Combined with paging, a sort operation resets the current page to the first page.

Only when no paging is used can this use case can be implemented on the client side of the application.

When a user navigates from a sorted list to an entity detail page, the notion of the sort order must be maintained in case the detail page provides paging at the instance-per-page level.

3.2.3 View a Whatever

Use Case Brief

For data items with more than just a few attributes, an entity details display is used to present all the attributes on a single screen, or indeed on multiple (tabbed) screens when there are many attributes involved. View a Whatever satisfies the goal of showing this entity details screen.

View a Whatever brings a single instance of Whatever in context. This is a result of:

- "Find a Whatever" matching only one single instance,
- Selecting a Whatever from "List Whatevers",
- "Filter Whatevers", matching only one instance, or
- Committing a newly created, copied or updated entity instance.

Note that there is no globally visible "View a Whatever" command because this would always require an identification of the Whatever to be displayed. Hence "View a Whatever…" would be an option, followed by a simple "Find a Whatever" to enter a unique identifier.

Success Guarantees

The instance details are displayed.

Points to Consider

A single instance of Whatever is in context.

In many cases, View a Whatever can be merged with "Update a Whatever". There is simply no read-only display of entity details in such situations. Look at the target audience (the use case actor) to determine when this is appropriate.

3.2.4 Next Whatever

Use Case Brief

As part of data management, users can filter or search a particular domain of data items. Once they have identified a subset of interest, they wish to step through the instances one by one. This is referred to as *paging* and includes typical commands

© MphasiS Corporation

such as Next, Previous, First and Last. In small domains, paging may also occur on the full set of data items.

Success Guarantees

The details of the next (previous, first, last) instance are displayed.

Points to Consider

Referring to the use case brief above, note that only one of the four navigation paths to "View a Whatever" includes an implicit set of instances to page through; this is only the case when one of the items on a list of whatevers is selected for drill-down. The other navigation paths have no implied notion of a record set – for these the full domain is the only context. Due to the absence of a record set there is also no upfront concept of an ordering in these cases.

3.3 Searching for Whatevers

There are two fundamentally different approaches to searching. One approach, referred to in this white paper as "Finding", uses fill-in forms to enter search crite-ria. The alternative approach is known as "Filtering". The system presents a list of values and the user selects one of the values Figure 5 below shows aclassical example of filtering.

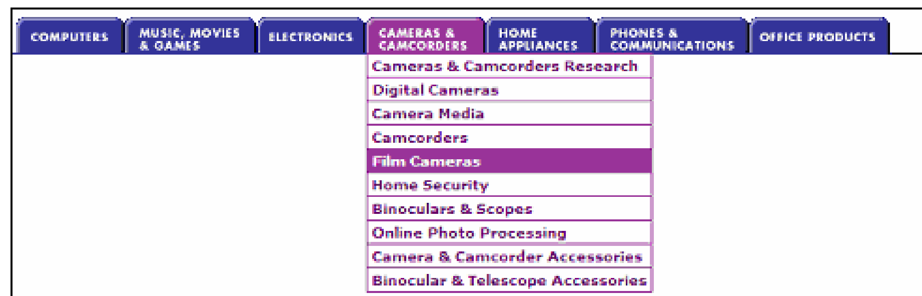

Figure 5 Catalog Browser at BestBuy

The big difference between finding and filtering is that a filter operation cannot result in an empty result set, as the values to choose from are derived from the actual data instances.

3.3.1 Find a Whatever

Use Case Brief

When there are many instances of a particular type of data, Manage a Whatever requires a search mechanism to locate an instance or instances of interest. This is even more relevant in situations where the entity does not provide any filtering features (see "Filter Whatevers" on page 14).

© MphasiS Corporation
MphasiS an HP company

Success Guarantees

One or more matching items are found.

Points to Consider

The alternative trigger for Find a Whatever occurs when an implementation allows the inline search for data values while working on some other task. Consider the case where an online order entry application allows adding order lines one by one. The user interface should allow users to search for product codes (by partial match on the code number or on the description) in the middle of the order creation process.

3.3.2 Filter Whatevers

Use Case Brief

Filtering is the complement of Find a Whatever. Rather than searching for a particular entity instance, "Filter Whatevers" implements a guided selection of entities. For a particular attribute, Filter Whatevers shows all the attribute values that are in use. The user selects one value of interest and as a result the system displays all matching Whatevers. The difference with Find a Whatever is that Find uses data entry, whereas Filter only uses selection from value sets.

Success Guarantees

The results are displayed.

Points to Consider

Filtering requires that the entity has enumerated attributes (i.e. attributes that form a closed set of values).

The elements to display are the attribute values currently in use (the ones that are referenced in the entity domain).

The result of filtering can be a single instance or a collection of instances, but never an empty set.

3.4 Extension Points

Sections 3.1 to 3.3 describe the baseline pattern language of Manage a Whatever. In this section we outline a standard approach to extending the pattern language.

3.4.1 Service a Whatever

Use Case Brief

The "Manage a Whatever" pattern focuses on CRUD-style tasks for application data. Many applications require *additional* tasks that apply to data . As long as such tasks are limited to a single instance of Whatever, this use case can be used as an extension point for implementation of these tasks.

A typical example of this is "mark as read" which applies to an email message. It changes the state of one particular email message. Another example is "check

© MphasiS Corporation
MPHASIS an HP company

availability" on an order line item. A situation that is not well covered by "Service a Whatever" is "find the difference between these two email messages", because it requires two email messages to be in context, not just one.

"Service a Whatever" can also be implemented as an extension of "List Whatevers", but the service being invoked would still apply to single instances (it would iterate over the set of instances that is selected).

Success Guarantees

The service is successfully completed on all instances.

Points to Consider

A single instance of Whatever is in context, or a set of target instances is selected in a list.

The typical scenario for "Service a Whatever" is that one or more instances of Whatever get selected, a command is applied to them (possibly transforming the Whatevers), and the same Whatever or list of Whatevers is displayed again. In many cases, the command does not require any additional input from the user and the whole sequence is completed with one click on a button.

Strictly speaking, this use case can cover some of the earlier use cases such as "Delete a Whatever", "Update a Whatever", "Sort a List of Whatevers" and "Copy a Whatever". We separated those commands out into their own use cases because they clearly are part of the core CRUD capabilities of Manage a Whatever.

There are two typical services that find their way in many occurrences of "Manage a Whatever". The first is "Print", which applies equally to a single Whatever and a list of Whatevers, although the output format may differ. The other service is "Export", typically used to pull data from a data source in a structured way, ready for import into some other application. It can be argued that these two commands should be specified through their own use cases, "Print a Whatever", "Print List of Whatevers" and "Export Whatevers".

3.5 Pattern Variations

While reading the use case analysis texts in this chapter you, as a business owner or requirements analyst, may have questions regarding the applicability of one or more of the use cases in a particular scenario you had in mind. That's perfectly acceptable – there will be many situations where the pattern shouldn't be deployed at full force. We shall identify some situations that call for pattern variations; both simplifications and extensions.

3.5.1 Limited Domain

A limited domain is a domain that is known to include only a handful of instances, ever. In a limited domain, there is still a need for Create, Update and Delete Use Cases. We're still managing some data, after all. We won't need Find or Filter. Find is pointless because the full domain list fits on a single screen, Filter is pointless. We probably won't need Copy, either.

© MphasiS Corporation
MPHASIS an HP company

3.5.2 Final Domain

In a Final Domain, entity instances never need to be updated. Examples of these are lists of country codes and currencies. New currencies may be introduced, but the code used for an existing currency never changes. This pattern variation hence does not need any Update and in most cases, Delete is also not applicable.

Note that Final Domain can be combined with Limited Domain, but that is not always the case. A good example is the Stock Trading Symbols domain. There are thousands of symbols, and they are all final. The "Manage Stock Symbols" pattern is of the final domain variety, but definitely not a limited domain.

Note that these domains are only final in the scope of the system under study. It may well be that updates and deletes are tolerated (perhaps using direct data editing facilities in the underlying RDBMS). A Final Domain simply means that Manage a Whatever does not cater for Update or Delete.

3.5.3 Unbrowsable Domain

In an Unbrowsable Domain, there are no attributes that are appropriate for filtering as there are no enumerated attributes. In the Library (see section 2.1 beginning on page 3), there are several of these:

- Subject
- Publisher
- Author
- Keyword
- Staff

Referring to Figure 4 on page 6, these entities have no arrows going out of them, only arrows pointing towards them. In reality, all limited domains are also unbrowsable domains. Some unbrowsable domains are not limited though.

3.5.4 Read-only Domain

This may sound contradictory in the context of data management, but in some situations the source of the data is not in scope of the application and no interfaces are available to modify data or create new instances. Examples of this scenario are Directory Services, Repositories and external systems that are Web-Service enabled but only for viewing.

A read-only domain is read-only in the context of the system under study, only. It could still be that there are other tools available to create, update or delete instances. It may even be a conscious decision to implement a domain as read-only and leave all data management tasks out of scope for the system under study.

A comparable situation arises when modifying the data source is in scope but not for the current user. The entitlements of a given user may simply not allow Create or Update scenarios.

In a Read-only Domain, all use cases apply except Create a New Whatever, Copy a Whatever, Update a Whatever and Delete a Whatever. Service a Whatever is probably not very useful, although still valid for query-style services such as Export.

© MphasiS Corporation

3.5.5 Cascaded Filtering

Cascaded filtering refers to the situation where the user filters one dimension first, and the result of her/his selection is another domain to filter by, rather than a list of matching entities. Figure 5 on page 13 showed an example of this. After selecting "Cameras & Camcorders", the system presents a dropdown menu with types of cameras to choose from – not a list of camera models to buy.

In the Library project, Publication has many enumerated dimensions, but only one of them is cascaded: the Subject – Category – Publication association.

Cascaded Filtering is an extension of the "Filter Whatevers" use case.

3.5.6 Master-detail Domain

A Master-detail domain refers to entities that are master in a master-detail relationship. In the Library, Publication is a master-detail domain with respect to Copy.

Because detail entities never exist outside the context of their master entity (they are part-of that master), some use cases for "Manage the Detail" are integrated with "Manage the Master".

Create a Detail – Because detail entities are always created in the context of their master, it makes sense to integrate Create a Detail with either View a Master or Update a Master. This makes the identification of the master entity implicit which is a good thing.
The Library project has Create a Copy integrated with Update a Publication. Because Update a Publication shares data entry screens with Create a New Publication, copies can be created as part of creating publications as well. This makes sense because a publication would never be added to the library if no copy was available in the first place.
Ticket reservation systems integrate Create a Seat Reservation with View a Show (or View a Flight, or whatever the tickets may be used for). The show isn't created together with the first seat reservations, so the integration with Create a Show does not make sense.
Depending on the complexity of Create a Detail the task may require a separate screen or the detail entity may be created on the same page where the data is present.

List & Filter Details – one way to filter details is through their master entity. In case the detail rows are not shown on the View Master page, there should at least be a way to navigate to the list of detail records. From the list of detail records, a single detail entity can be selected, leading to…

View a Detail – this now shows all the attributes of one detail entity, and possibly some of the attributes of the master. From here, Update, Copy and Delete lead to the respective use cases.

© MphasiS Corporation

MphasiS
an HP company

4 MphasiS Case Study - Web-Enabled Order Visibility

4.1 Business Objectives

Our client is a global manufacturer of consumer packaged goods (CPG). The company uses SAP R/3 Sales & Distribution for its sales and logistics processes. To better service their customers, our client wanted to provide web access to all customer orders as currently visible in the SAP system.

4.2 Project Overview

The business requirements related to order visibility can be summarized as follows:

[1] Orders must be selected based on one or more of the following attributes: Customer ID, Order Entry Date, Vessel Sailing Date, Order ID, Purchase Order Number and Product Code. Both single values and ranges must be allowed where applicable.

[2] Order details include: Purchase Order Number, Order Entry Date, Bill-to Details, Ship-to Details, Product details including product code and required as well as actual quantities, Payment Terms, Order Value, Order Status

[3] Logistics details include: Carrier Name, Vessel Name, Planned and actual sailing date of vessel, Planned and actual arrival date of vessel

[4] Order status details for the following statuses: Held, Planned, Firm Planned, Produced stock, Picked, Loaded, Dispatched, Invoiced, Vessel Sail, Port of Arrival, Paid

[5] For each order, a full list of associated documents must be shown. The documents themselves need not be made available.

[6] The breakdown of an SAP Sales Order across multiple shipments and in turn across multiple deliveries must be shown

[7] Users can print orders in a convenient A4 format and download order data in CSV or XML format

[8] There will be both internal and external users

[9] Roles and profiles should be established to define access to order information and detail visibility

[10] User access must be bound to roles

[11] Inactive users should be blocked after 30 days without access.

[12] Internal users must have access to all details visible to the customer

[13] User profiles must allow that one user can be associated with more than one sold-to customer.

[14] Users must have the option to choose one of three foreign languages in addition to English. All static text must be available in these four languages, but SAP data shall not be translated.

[15] Administrators must have visibility of site activity. Log data should be captured by user name, customer account number, date & time, functionality accessed and details of the data access.

© MphasiS Corporation

4.3 MphasiS Approach

Candidate Entities for Manage a Whatever

Based on these requirements, we identified the following managed entities:

Order – This represents the primary object of interest for the application. It is purely read-only data. In terms of the pattern language, this is a read-only domain (see section 3.5.4).

Log – This domain is limited to system administrators, exclusively. It is invisible to all other users. The application itself is the only party entitled to add entries to the log. There are no updates to existing entries. This is a final domain (see section 3.5.2). The system administrator will have one goal related to deleting log entries, this is log truncation.

User – Application users are actively managed by the system administrator. Each user has access to his/her own profile for maintenance of preferences (language). Users are master entities with respect to Accounts and with respect to Roles. Beyond the system administrator, several other system users are entitled to register customers as user.

Role – Roles identify entitlement profiles. Each role refers to a collection of entitlements (Roles is master in a master/detail relationship with Entitlement). Administrators can create roles and alter the entitlements for each role.

Language – The requirements call for multiple languages. Users can specify their preferred language, administrators can add and remove languages.

Account – These are the distinct customers as available in SAP. They are referenced in this application but not actively managed in any way.

Unmanaged Entities

Beyond the managed entities, we also identified data entities that are used but not actively managed. These data entities must be managed through some other means, most likely the standard data management GUI of the underlying database management system.

The following unmanaged entities were identified:

Localized Message Catalog – These are all the messages of the application, translated into a target language.

Message – These are the definitions of the system messages. Each definition includes a message class such as ‘button label’, ‘tool tip’, ‘window title’, ‘help text’ etc. Other attributes include maximum string length and contextual description.

Message Class – This is the list of message classes to choose from.

Dialog – Each application page (or dialog) includes a set of messages. This master/detail relationship is modeled in the Dialog entity.

Entitlement – These are the base entitlements the system has to offer. They are mapped one-on-one to individual pieces of system functionality. Roles are used to provide access to these functions.

© MphasiS Corporation

4.4 Application of the Pattern Language

Here, we show how the pattern language is applied to two domains. *Order* is the primary area of interest for this application, hence there are numerous requirements that relate to this. *User* management is definitely not a business goal, although there are still several requirements associated with users.

Use Case Analysis for Order

From the requirements, we found the following occurrence of Manage a Whatever:

- Order is a read-only domain hence no use cases for create, copy, delete or update are required.
- From requirement [1], *Find an Order* is part of the pattern. Because of requirement [1], there is an implicit requirement for the display of search results in a list format. This brings *List Orders* in scope. As there can be many orders, the implementation must include paging.
- From requirements [2]-[6], *View an Order* is in scope. Potentially more than one page is required to show all the order details; candidate sub-use cases are: *View Order Summary, View Order Shipments, View Order Deliveries, View Order Documents*.
- Although not specifically addressed, users must indicate for which account they wish to see orders. This brings *Filter Orders by Account* in scope.
- Requirement [7] is covered by *Service Orders*. In the given situation, there will be four appearances of this use case: *Print this Order, Download this Order, Print Selected Orders and Download Selected Orders*.

Order Use Cases

The resulting Use Cases for Order are:

[1] List Orders
[2] Sort a List of Orders
[3] Print Selected Orders
[4] Download Selected Orders
[5] View an Order
[6] Next Order
[7] Print an Order
[8] Download an Order
[9] Find an Order
[10] Filter Orders by Account

Use Case Analysis for User

The requirements for the User domain are much less specific, when compared to Order above. This is very common, as the management of application users is not a business goal but rather a necessary evil.

Most of the user-related use cases are captured in requirements [8]-[14]. The typical use cases include:

- *Add a User* which is tied to the administrator role. *Copy a User* to minimize the effort required to set up user access details.

© MphasiS Corporation

- *Update a User* comes in three distinct variations: the system administrator can modify any user detail, assigned roles can associate users with Accounts (*Entitle a User*), and users can set their own preferences (*Specify my Preferences*).
- The Roles of a User form a master/detail relationship. This is a pattern variation, discussed in section 3.5.6. For the detail side (Roles), additional use cases will come into play, most of which relate to defining the roles. One use case refers to User, which is *Add a Role to a User.* This is exactly what *Entitle a User* does.
- *Delete a User* is never mentioned but certainly needed as part of the provisioning process.
- *Find a User* is also never mentioned but given the expected user volume (hundreds of registered users), some form of selection is definitely in order.
- User listing and viewing are again never mentioned but certainly required. *View a User* can be merged with Update a User. *List Users* requires full paging support.
- The following *services* apply to User: *Delete a User*, *Activate a User* (see requirement [11]), *De-activate a User*, *Download User Data*, *Print User Data*. These services shall be made available on the user list page as well as the user edit page.

User Use Cases

The resulting Use Cases for User are:

[1] Create a New User
[2] Copy a User
[3] Update a User
[4] Entitle a User (aka Add a Role to a User)
[5] Specify my Preferences
[6] Delete a User
[7] List Users
[8] Sort a List of Users
[9] Activate Selected Users
[10] De-activate Selected Users
[11] Print Selected User Details
[12] Download Selected User Details
[13] Print this User Details
[14] Download this User Details

4.5 Bottom Line

The structured approach to identification of candidate domains and associated use cases simplifies the process of requirements gathering. With the use cases skeleton in place, any follow-on requirements elicitation can focus immediately on the details of the individual use cases.

For the User domain, we have found that *Delete a User*, *Find Users* and *View a User* are not at all mentioned in the requirements. Applying the Manage a Whatever pattern languages has led to an early heads-up on this omission.

© MphasiS Corporation

For the other domains, a similar approach has been followed. As a result of that, a substantial list of use cases has been made available very early in the business requirements gathering process, rather than at the very end. With the candidate use case list available, the discussions with the client could focus on project scope and alternative solutions to out-of-scope use cases.

© MphasiS Corporation

5 Conclusions

In Chapter 2, "Business Requirements", we have identified a business requirement for the management of data items. The requirement is often implicit and sometimes well-hidden. We have seen how the need for data often leads to a requirement for *managed* data . In this chapter, we have identified a number of distinct use cases that together cover the generic requirements for data management.

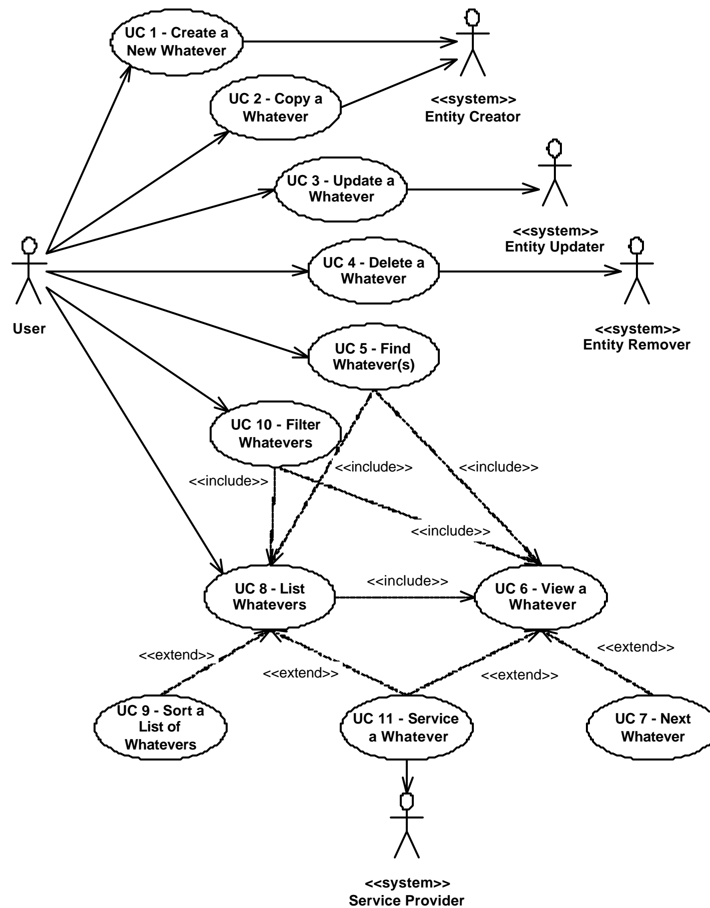

Figure 6 Use Case Diagram for Manage a Whatever

The use cases are:

- Create a New Whatever
- Copy a Whatever
- Update a Whatever
- Delete a Whatever

© MphasiS Corporation
MPHASIS an HP company

- Find a Whatever
- View a Whatever
- Next Whatever
- List Whatevers
- Sort a List of Whatevers
- Filter Whatevers
- Service a Whatever

These use cases are captured in a Use Case Diagram in Figure 6.

While discussing these use cases, we have discovered relationships between them as well. These relationships are actually part of the Manage a Whatever pattern – they apply universally, irrespective of what the Whatever is about.

Some of these relations refer to navigability – Next Whatever only makes sense when the result of View a Whatever is shown. Other relationships tell us which page to display as a result of applying a command. Filter Whatevers end with either List Whatevers or with View a Whatever. In a follow-on white paper, we will analyse these relationships and define a *page flow pattern language* for data management.

We have also identified a number of variations on the general pattern. Some of these extend the basic pattern, others simplify it.

During the Requirements Analysis phase of any project of significance, the material presented in chapters 2 and 3 can be used to structure the analysis and quickly identify the scope:

Identify all candidate data items. The result of this step is a list of managed entities.

For each candidate, identify which variation(s) of the Manage a Whatever pattern apply. This requires some analysis in terms of expected volume of instances and complexity of the entities. Outcome of this step should be, for each entity, a short specification that identifies the use cases to be implemented. The minimal approach to this would be a “Requirements Matrix”, as per Table 2 on the next page.

© MphasiS Corporation

Entity	Variation	Create	Copy	Update	Delete	Find	View	Next	List	Sort	Filter	Service
Publication	Master-Detail	✓	-	✓	✓	✓	✓	✓	✓	✓	✓	Print
Subject	Limited, Final	✓	-	-	-	-	✓	-	✓	✓	-	-
Category	Cascading	✓	-	✓	✓	-	-	-	✓	✓	✓	-
Publisher	Unbrowsable	✓	-	✓	✓	✓	-	-	✓	✓	-	-
Keyword	Limited	✓	-	✓	✓	-	-	-	✓	✓	-	-
Author	Unbrowsable	✓	-	✓	✓	✓	-	-	✓	✓	-	-
Copy	regular	✓	-	✓	✓	✓	✓	✓	✓	✓	✓	Take out
Staff	Read-only	-	-	-	-	✓	✓	✓	✓	✓	✓	-

Table 2 Sample Requirements Matrix for Library

© MphasiS Corporation
MPHASIS an HP company

Appendix A - References

[1] "Writing Effective Use Cases", Alistair Cockburn, Addison-Wesley, 2001. ISBN 0-201-70225-8

[2] "Designing Easy-to-use Websites", Vanessa Donnelly, Addison-Wesley, 2001. ISBN 0-201-67468-8

[3] "Designing Data-Intensive Web Applications", Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, Maristella Matera, Morgan Kaufmann Publishers, 2003. ISBN 1-558-60843-5

[4] "Web Database Applications with PHP & MySQL – Building Effective Database-Driven Web Sites", Hugh E. Williams and Davis Lane, O'Reilly 2002. ISBN 0-596-00041-3

[5] "Web Application Design Handbook : Best Practices for Web-Based Software", Susan L. Fowler and Victor R. Stanwick, Morgan Kaufmann Publishers, 2004. ISBN 1-558-60752-8

© MphasiS Corporation