Aware IM

Version 8.5

Configuration Case Study
Library Application

CONTENTS

1 Introduction

The purpose of this document is to show how to configure real world business applications using ***Aware IM***. The document aims to serve as a studying and reference material for anyone who wants to configure business applications using ***Aware IM*** – it contains many practical examples, configuration and operation patterns and demonstrates the key features of ***Aware IM***.

The document has a format of a case study – it explains how to design and configure a particular business application (the software system to operate a public library). It covers all aspects of the application creation – from requirements to configuration and testing.

The document should be read in conjunction with the ***Aware IM*** documentation – in particular, the Aware IM User Guide and Aware IM Rule Language Reference.

The electronic version of the document in PDF format can be found in the “docs” directory of your Aware IM installation.

2 Versions of the Library Application

There are two versions of the Library application available for you to play with – the classic version (LibraryClassic.bsv) and the modern version (Library.bsv). Both versions have the same business objects and rules, but the user interface of the modern version is more contemporary. This document describes the classic version, so if you want to follow the document you should load LibraryClassic.bsv.

Once you are comfortable with the classic version you can load the modern version and check out the differences in the User Interface. The main difference is that queries on items and members that used to return a grid of items or members have been replaced in favour of “custom queries” that rely more heavily on HTML and CSS. Consequently many operations that used to be invoked from forms (such as borrowing an item) are called from custom queries using Javascript in the modern version.

3 Library Application

3.1 Overview

The example that we will look at in this document is the application to manage a public library. The system that we are going to configure will keep track of the library

resources, such as books, CD's and videos and allow its members to borrow and reserve the library items. It will also manage information about library members keeping track of their activity and payments and registering any communication between members and the library. Members will be able to access the library system online to check their outstanding loans, reservation and fees, make online reservations and credit card payments.

In the following sections we will show how to design and configure such a system using ***Aware IM***. Note that the complete configuration of the library system can be found in the "samples" directory of your Aware IM installation. You can use this configuration as reference when reading this document.

First we will list the requirements to the Library System so that we know exactly what system we are going to build.

3.2 Requirements

This section states, from the management point of view, what the Library System should be able to do. Note that the requirements stated here are in no way affected by the subsequent configuration details and are not adjusted to the features and capabilities of ***Aware IM***.

3.2.1 Items

1. Library item, or simply item, is a book, compact disk, video tape or any other resource that library members can borrow.
2. An item has a title and author(s)
3. Each item shall have a code, a short alphanumeric label uniquely identifying the item in the library.
4. It shall be possible to define an additional description of an item if required.
5. An item is kept in one of the well-known locations (usually inside a library but can be outside as well). From time to time some or all of items can be moved from one location to another.
6. An item can be of the types detailed in the table below. Each type indicates how many items of this type can be borrowed by a member at any one time, maximum loan period in days and the number of times an item can be renewed:

Type	**Max number that can be borrowed**	**Loan period**	**Max renewals**
Book	10	28 days	2
Video	2	14 days	1
DVD	2	14 days	1
Magazine	4	14 days	1
CD	4	28 days	2
CD-ROM	4	28 days	2

Cassette	4	28 days	2
Community language book	4	28 days	2
Literacy item	5	28 days	2
Talking book	10	42 days	1

7. Users shall be able to see the photograph of the item where available.
8. The library may withdraw the existing item from use. The withdrawn items are retained by the library but are not available for borrowing any more.

3.2.2 Members

1. Only registered members can borrow items from the library.
2. The following member information shall be registered:
   - Name.
   - Address.
   - E-mail address, optional.
   - Age.
   - Gender.
   - Library-assigned 8-digit leading zero-padded membership number.
3. For certain violations the library management can suspend the member's ability to borrow items. The management can reverse the suspension.

3.2.3 Borrowing Items

1. Unless suspended a member can borrow items from the library.
2. The maximum number of borrowed items cannot exceed 10 at any given time. From time to time the management can change this number, as well as item type-specific numbers. For example, during certain periods, such as school holidays, the numbers can be increased.
3. The loan duration is determined by the item type (see the item type table).
4. An item can be borrowed on any day.
5. The loan due date is determined based on the lent date and the loan duration.
6. The number of borrowed items of the same type, at any given time, cannot exceed the maximum specified for the type (see the item type table).
7. A member can renew the loan up to the maximum number of times as determined by the item type (see the item type table) and provided there are no reservations on the item.
8. If the borrowed item is not returned on the due day then on the next day:
   a) The member is charged $1.00 overdue fee.
   b) The member is sent an e-mail if the e-mail address is available or an ordinary letter otherwise, informing the member of the overdue loans including the details of the loan.
   c) The fee and reminder letter are issued again with a 7-day interval until the borrowed item is returned.
9. Upon item's return the return date is registered and the loan is closed.
10. It shall be possible to produce at any time a printable loan statement showing the member details including currently borrowed items with indication of the following:
    - Item code.

- Item title.
- Lent date.
- Due date.

11. Details of all loans, whether current or past, are kept for reference purposes.

3.2.4 Reserving items

1. If an item is currently on loan a member can make a reservation for the item.
2. The reservation fee $2.00 and is charged to the member at the time the reservation is made. The fee is not refundable.
3. A member cannot reserve an item that is currently on loan to the member.
4. Once the reserved item is returned to the library:
   a) The member who reserved the item is notified by e-mail if the address is available or by ordinary letter otherwise. The e-mail/letter contains the details of the reservation, including the title and the reservation expiry date.
   b) A reservation remains open until the reserving member borrows the item or 7 days elapses in which case the reservation expires.
   c) If a reservation expires the item becomes available for borrowing to the next reserving member (who is notified accordingly) or to anyone if there are no more reservations.
5. If there is more than one reservation such reservations are processed in the order they are made.
6. A reservation can be cancelled by a member's request.
7. All reservations regardless of their status are kept for reference purposes.

3.2.5 Fees

1. The library can charge its members fees for certain services or in certain events.
2. There are several predefined types of services/events.
3. When a fee is charged the system shall record the fee amount, date and the description of what the fee is for.
4. Once the fee is charged it increases the outstanding fees of the member by the fee amount.
5. Library management can remove a fee from the member in which case the outstanding fee amount should be decreased by the fee amount.

3.2.6 Payments

1. A member can pay all or part of the outstanding fees
2. A member can pay online to the account of the Library by credit card. The system shall record the date and amount of the online payment.
3. A member can also pay by other (non-electronic) means, such as cheque, money order etc. It shall be possible to make an electronic record of the non-electronic payment in the system registering the amount of payment, the date of payment and the description of what the payment is for.
4. Once a payment is made the outstanding fees are decreased by the payment amount.
5. When a payment is made (both online or offline) a printed summary of the payment shall be produced for the member showing the following:

- Member name.
- Date.
- Amount.
- Updated outstanding fees.

3.2.7 Communication with members

1. It shall be possible to record and store electronically the details of all forms of relevant communication between the library and its staff with library members. In particular:
   - E-mails to members
   - Letters to members
   - E-mails from members
   - Faxes to members
   - Faxes from members
   - Results of phone calls
   - Results of discussions
2. The results of communication with a member shall be easily accessible on request

3.2.8 Information access and activity restrictions

1. Library operators are allowed to see all information and perform all actions with the exception of modifying the details of the table in 3.2.1.6.
2. Only the library manager can change details of the table in 3.2.1.6.
3. Members can change their personal details and view their loan, reservation, fee and payment history. The only operations a member can perform are to reserve an item or to renew a loan, subject to the applicable rules. A member cannot see any information related to other members.
4. A guest of the library (any person not registered as library member) is allowed to browse and search the library collection of items, but is not allowed to perform any other operation or change any of the details.

3.2.9 Internet access to library information

1. Members and guests should be able to access library information from outside of the library via the Internet using a standard Internet browser. No special software should be required to be downloaded or installed on the user's computer.
2. New library users should be able to register themselves as members via the Internet.
3. To access the library information members must login and their user name and password should be verified against the registered details.
4. Guests should not go through login process and can directly access library resources, subject to the restrictions for guests.

4 Configuration of the Library Application

This section will show you how to configure the library application in ***Aware IM*** in a sequence of relatively simple and straightforward steps. These steps are described in detail in the “Configuration Guidelines” section in the Aware IM User Guide. However, the steps and the order in which they need to be performed are so important that it is worth re-iterating them here:

1. Identify business elements of your system (business objects) and their interrelationships.
2. Identify business rules according to which these business objects behave. Do not worry about the user interface at this stage - let ***Aware IM*** generate the default user interface.
3. Identify scenarios of user interaction with the system. Configure main system processes and operations.
4. Test the core application.
5. Fine tune the application:
    - Add access control
    - Add documents and reports
    - Improve system operations
    - Improve the user interface – customize forms and application visual appearance.
6. Test the final application.

We will cover all these steps in detail in the following sections.

4.1 Business Objects and Relationships

The first and most important step in designing a system is to identify business objects and business rules of the system. If you get this right everything else will be relatively straightforward.

You need to carefully look at the requirements and identify the main business elements that the system deals with (business objects), relationships between them and rules according to which these objects behave. At this stage it is very important that you do not worry too much about how the system will look to the user and how the user will interact with the system.

So we start by identifying business objects and relationships between them. For clarity we will temporarily not consider some of the requirements – we will come back to them later after we have configured the core of the system. The requirements that we will not consider at this stage are the management of communication with library members (3.2.7) and the credit card payments (3.2.6.2).

Some of the business objects are quite straightforward – if we carefully study the requirements we can identify the following business objects:

- Item

- Member
- Loan
- Reservation
- Fee
- Payment

Let us take these objects as a starting point and see if there are any relationships between these objects. One of the straightforward relationships is the relationship between Member and Loan – a member can have multiple loans; but a loan is always for some particular member. So there is a multiple relationship between Member and Loan and a single relationship between Loan and Member.

Similarly there is a clear relationship between Loan and Item – a Loan knows which item has been borrowed; an Item knows which loan is registered against it (if any) or which loans have been registered in the past. So there is a single relationship between Loan and Item and a multiple relationship between Item and Loan.

Is there a relationship between Item and Member? Yes there is, since an item should know which member has borrowed it or which members have borrowed it in the past. However, the Item object already has the relationship with the Loan object and the Loan object – with the Member object, so we already have a relationship between the Item and Member objects – through the Loan object. Therefore the relationship between Item and Member is indirect. In our exercise of identifying relationships we should only focus on direct relationships.

Looking at other objects we can conclude that there are the following direct relationships:

- Item object is related to:
    - Loan (an Item can have multiple Loans – one current and several past)
    - Reservation (an Item can have multiple Reservations – both current and past)
- Member object is related to:
    - Loan (a Member can have multiple Loans – both current and past)
    - Reservation (a Member can have multiple Reservations – both current and past)
    - Fee (these are Fees the Member have been charged – both current and past)
    - Payment (these are Payments that the Member has made)
- Loan object is related to:
    - Member (this is the single Member that borrowed an item)
    - Item (this is the single Item borrowed)
- Reservation object is related to:
    - Member (this is the single Member who made the Reservation)
    - Item (this is the single Item that has been reserved)
- Fee object is related to
    - Member (this is the single Member who was charged)
- Payment object is related to:

- Member (this is the single Member who made the payment)

The above business objects and relationships are relatively obvious. Another business object is slightly harder to identify.

If we look at the requirements in section 3.2.1 we will notice that items can be of different types. What is the type of the item? It is clearly something that belongs entirely to the item, so it could be the attribute of the Item object. However, if we look at the table in 3.2.1.6 we will notice that there is some extra information associated with the item type, i.e. maximum number of loans, loan period and maximum number of renewals. Where does this information belong? If item type were an attribute of the Item object then this information would have to belong to the Item object as well. In this case every instance of the Item object would store the values of maximum number of loans etc. This is very impractical since according to the requirement in 3.2.3.2 every now and then the library manager needs to change this information. If this information were stored in every instance of the Item object the manager would have to change it in every item that exists in the library instead of changing these values in one place!

This means that the ItemType is an object in its own right and that the maximum number of loans, loan period and maximum number of renewals are attributes of this object. If a manager wants to change the values of these attributes she should just find the ItemType objects and change the values there. The Item object should therefore be related to the single ItemType object that it belongs to.

Therefore we add the ItemType object to our list of objects and the following relationships to our list of relationships:

- Item object is related to
  - ItemType (this is the single ItemType that it belongs to)
- ItemType object is related to
  - Item object (these are multiple Items that belong to this ItemType)

4.1.1 Attributes of Business Objects

Now that we have defined business objects and relationships we can define attributes of each business object. There are two types of attributes –simple attributes (text, number, date, picture etc) and those representing relationships or *references* (see also the “Reference Attributes” section in the Aware IM User Guide).

Values of some attributes are entered by users, whereas values of other attributes are always calculated by the system and never entered by users. The latter attributes are called *calculated* attributes.

By carefully studying the requirements we can identify the following attributes of the objects:

Item Object:

- *Title* – text

- *Author* – text
- *Code* – text
- *Description* – text
- *Picture* – image
- *Location* – text
- *InternalStatus* – text (with possible values “Released” and “Withdrawn”)

Now reference attributes (see the relationships of the Item object):
- *Type* (a reference to the single ItemType object that the Item belongs to)
- *Loans* (a list of Loan objects for the item, calculated)
- *Reservations* (a list of Reservation objects for the item, calculated)

ItemType Object:
- *Name* – text (for example, Book, CD, Video etc)
- *MaxBorrowNumber* – number
- *MaxLoanPeriod* – number
- *MaxRenewals* – number

Member Object:
- *FirstName* – text
- *LastName* – text
- *Address* – text
- *DateOfBirth* – date
- *Gender* – text (Male or Female)
- *EmailAddress* – text
- *MemberNumber* – number (unique number associated with a member - calculated)
- *Photo* - picture
- *Status* (“Active” or “Suspended”)

References:
- *Loans* (a list of Loan objects, calculated)
- *Reservations* (a list of Reservation objects, calculated)
- *Fees* (a list of Fee objects, calculated)
- *Payments* (a list of Payment objects, calculated)

Loan Object:
- *DueDate* – date (the date the loan is or was due)
- *ReturnDate* – date (the date the loan was returned - calculated)
- *Status* – text (New, Current, Past - calculated)

References:
- *Member* (reference to a single Member object, calculated)
- *Item* (reference to a single Item object, calculated)

Reservation Object:
- *ReservedOn* – timestamp (the date and time when the reservation was made, calculated)

- *BorrowedOn* – timestamp (the date and time the reservation was borrowed, calculated)
- *ItemOfferedOn* – timestamp (the date and time the reservation was offered, calculated)
- *Status* – text, calculated (New, Waiting – if reservation is waiting for item to be returned, Offered – if reservation has been offered but not taken up, Converted – if reservation has been taken up, Cancelled – if reservation has been cancelled by owner, Expired – if reservation has expired)

References:

- *Item* (a reference to a single item that has been reserved, calculated)
- *Member* (a reference to a single member who made the reservation, calculated)

Fee Object:

- *Amount* – number
- *Description* – text
- *AppliedOn* – date (the date the fee was charged, calculated)

References:

- *Member* (reference to a single member who was charged, calculated)

Payment Object:

- *Amount* – number
- *Description* – text
- *AppliedOn* – date (the date the payment was made, calculated)
- *Status* – text (New or Applied)

References:

- *Member* (reference to a single member who made a payment, calculated)

In addition we will introduce a few attributes that will display summarized information to the user:

Item Object:

- *Available* – this is the flag that indicates whether the item is available. In principle, this information can be derived from the information about loans associated with the item. It is, however, very convenient to immediately show to the user whether the item is available (both in queries on item or in the item form)
- *CurrentReservation* – this is a reference to the reservation that is the first in the queue of reservations. We define this attribute purely for convenience to avoid calculating it every time it is needed (it will be needed by a number of rules).

Member Object:

- *OutstandingCharges* – this is the number indicating whether the member owes something to the library. Again, in principle, this information can be derived from the information about fees and payments, however, it is very convenient to show this number to the user.

4.1.2 Configuration of Business Objects

Now that we have determined the business objects, their relationships and attributes we will explain how to enter this information into ***Aware IM***. All configuration work in ***Aware IM*** is done in the Configuration Tool. The Configuration Tool can be started from the ***Aware IM*** Control Panel by clicking on the "Start Configurator" button.

After the Configuration Tool starts it displays the tree with all elements of the configuration and the working area where you define the properties of the configuration elements (see also the "Overview of the Configuration Tool" section in the Aware IM User Guide).

Configuration of the application is performed within a *business space*. For our library application the business space is called "Library". You need to specify this name when you start the Control Panel for the first time or create a new business space by selecting the File/New item of the Configuration Tool menu"[1].

Inside the business space configuration elements of the system are contained in *business space versions*. When a business space is initially created it contains one business space version (which has some pre-defined elements). Configuration work usually begins with the configuration of the business space specific elements in this version – see also the "Working with the Business Space Versions" section in the Aware IM User Guide.

So we start our configuration work by creating business objects in the default business space version (usually it is version 1.1 in the business space). Select this business space version and expand its node in the tree – you will see all configuration concepts available in ***Aware IM*** – business objects, processes, queries etc. We are currently interested in business objects. Right click on the "Business Objects" node and select "File/New" from the menu (or right click and select New from the popup menu). The working area will show the business objects editor.

Defining ItemType Object.

Let us start by defining the ItemType object. First we need to specify the name of the business object – we enter "ItemType" into the "Name" property of the object displayed in the list of properties on the right. Then we need to define the attributes of the object. We click on the  icon and specify the name of the attribute "Name" and select its type "Plain Text". The new attribute appears in the list of attributes and its properties can be changed in the list of properties on the right. This attribute is mandatory and so we tick the "Required" checkbox. We can also provide some typical values for users to choose from, like "Book", "CD", "Video" etc. We can specify them as "choices". So we click on the icon next to the "Choices" property and specify these values in the table. We must also tick "Other values allowed" checkbox to indicate that users may not only choose from the list of typical values but also type in the values of their own.

[1] If you have run the sample application that comes with ***Aware IM*** the Library business space will have been created automatically by ***Aware IM***.

In a similar fashion we define other attributes of the ItemType object – “MaxBorrowNumber”, “MaxLoanPeriod” and “MaxRenewals”. The only difference with the “Name” attribute is that these attributes are of the “Number” type and do not have choices of typical values. Once we have finished adding the attributes we can click on the “Save” button in the Configuration Tool toolbar (or select File/Save from the menu(to indicate that the business object is ready (we will accept the default values for all other properties of the object). After you click on the “Save” button you will see the ItemType object in the Elements Tree.

Defining Item Object.

Now we can define the Item object. We define the “Title”, “Author”,”Code”, “Location”, “Picture”, “Description” and “InternalStatus” attributes in very much the same way as we defined the attributes of the ItemType object. The type of the Picture attribute is “Picture” and we will also define two possible choices for the “InternalStatus” attribute – “Released” and “Withdrawn”. This time we will leave the “Other values allowed” checkbox un-ticked to indicate that “Released” and “Withdrawn” are the only possible values of the attribute.

When defining the “Available” attribute we will also tick the “Calculated” checkbox to indicate that the attribute is calculated by the system. The type of this attribute is “Yes/No”.

Defining a Reference Attribute.

Now we can define our first reference attribute – “Type”, which is the reference to the ItemType object. We start in the usual manner and define the name of the new attribute “Type” and we select “ItemType” as the type of the attribute. Note that the objects that we have defined already appear in the list of attribute types – thus we can define a reference attribute by using the defined objects as an attribute type. Now we need to define other properties of the reference attribute – we tick the “Required” checkbox to indicate that an Item cannot live without an associated ItemType and we leave “Multiple allowed” checkbox un-ticked to indicate that the Item object may refer only to one instance of the ItemType object.

We will define a matching attribute in the ItemType object. We will click on the “Matching Data” property and in the dialog that appears we will select the “Create” radio button and give a name to the matching attribute (“Items”). Once we save the Item object the attribute with this name will be automatically created in the ItemType object. This will create the relationship between the ItemType object and the Item object (the ItemType object would know all Items that belong to it). The beauty of this approach is that when we create a particular instance of the Item object in the Operation Mode and link it with a particular instance of the ItemType object, the instance of the ItemType object would automatically add the newly created Item to its list of Item objects that belong to this item type. This means that when you navigate to the ItemType object you will see all instances of the Item object that belong to this type and you can navigate to any of them! It is generally a good idea to be able to see the relationship from instances of both objects; it is also handy to be able to navigate to the related objects from both sides of the relationship. The “matching attribute” feature allows us to do precisely that.

Here we can also specify the type of the relationship between the Item object and the ItemType object – we will explain later what the relationship type is for. For now we will leave the relationship type as “Peer”.

Relationship types and matching attributes are described in detail in the “Reference Attributes” section of the Aware IM User Manual.

We can now save the Item object by clicking on the “Save” button in the toolbar or selecting File/Save in the Configuration Tool menu.

Note that when defining reference attributes we did not have to define extra database tables, foreign keys and link the tables, as most database systems would require you to do. ***Aware IM*** creates all the required database tables and links between tables automatically allowing you to simply describe relationships between your business objects.

Defining Loan Object

Note that when defining the Item object we did not define references to the Loan and Member objects. This is because we have not defined these objects yet, so they are not present in the list of attribute types. We need to define these objects first and then return to the Item object to define the missing references (in fact, we will be able to bypass this step by creating matching attributes – see below).

Let us define the Loan object. We define the “DueDate”, “LoanDate”, “ReturnDate”, “RenewalCount” and “Status” attributes first. All these attributes except the “DueDate” should be calculated automatically by the system and are, therefore, marked as ‘calculated’. The “DueDate” attribute will be in most cases calculated by the system, however, we want to let the operator manually change the calculated value as well. So we do not mark the “DueDate” as “calculated” attribute. We will set the initial value of the “LoanDate” attribute as “`CURRENT_DATE`” to make sure that when the Loan object is created the value of the “LoanDate” attribute is automatically set to the current date.

Next we will define the reference to the Item object. We specify the name for the new attribute “Item” and select “Item” from the list of attribute types (we have already defined the Item object so it is available in the list of attribute types). We leave the “Multiple allowed” checkbox un-ticked as the loan refers only to one item. We also select the “Create” radio button on the “Matching Data” dialog and enter “Loans” as the name of the matching attribute in the Item object.

Now we click on the “Save” button to create the Loan object. Note that we did not define the reference to the Member object because we have not defined the Member object yet.

We can now go back to the Item object and see that the “Loans” attribute has been automatically created in this object (to edit the existing object we right click on the object’s node in the Elements Tree and select “Edit” from the pop-up menu or just

double click on the object). Let us have a look at this attribute. We select it in the attributes panel and look at its properties. We can see that the attribute was created with "Multiple allowed" checked – this is what we want in this case since the Item object can refer to multiple loans. If this wasn't the case we could change the value of the "Multiple allowed" checkbox. We can also see that the matching attribute for the "Loans" attribute is "Item" (the attribute that we have just created in the Loan object).

Defining Member Object

Now we can define the Member object. We define all simple attributes as usual and we also define the only reference we can define at this stage – the attribute with the name "Loans" of the type "Loan". This will be the attribute where "Multiple allowed" is checked since a member can have multiple loans. We also create the matching attribute on the Loan object called "Member". After we create the Member object we go back to the Loan object and change its automatically created "Member" attribute to turn off the "Multiple allowed" flag since a loan can refer only to one member.

Defining Other Objects

In a similar fashion we define other objects:

- Reservation with two references to Member (multiple not allowed; matching attribute in Member – "Reservations") and Item (multiple not allowed; matching attribute in Item – "Reservation").
- Fee and Payment with a reference to its Member object (multiple not allowed; matching attributes in Member – "Fees" and "Payments" respectively). For Fee and Payment objects we will choose to make their relationship with the Member object as a relationship of an "owner" (parent) and "owned by" (child), rather than "peer". The "peer" relationship indicates that the instances of two objects can exist independently of each other, whereas the "Owner of" relationships indicates that the instances of the referred object may not exist without the instance of its parent, so if the instance of the parent is deleted the instances of the child must be deleted as well (we will see the examples of the "Owner of" and "Owned by" relationship later). Fees and payments do not make much sense without the member they belong to – as soon as this member is deleted the corresponding fees and payments must be deleted as well. Therefore we select the relationship type for the Member attribute of both Fee and Payment object as "Owned by" in the dialog of the "Matching Data" property[2].

We have now created most business objects and their attributes. We will be creating a few more objects and attributes as we go along, but the core is there already. We now need to define business rules.

[2] Actually we could make the Member object a parent of not only fees and payments but also loans and reservations. However, with loans and reservations the situation is slightly more complicated because you could say that they are children not only of the Member object but also of the Item object. We could create two parents for these objects – for simplicity though we will make all these objects independent for now.

4.2 Business Rules

Now, that we have created our business objects and their relationships, the next step is to define business rules that will govern the operation of the library application.

When defining business rules we should look at every business object in isolation and identify the rules according to which this object "lives". It is important that we focus on a particular object and look at the world from this object's point of view only. This means that we should not be too concerned with what is going on around this object in the system (see also the "Business Rules as Carriers of Business Logic" and "Configuration Guidelines" sections in the Aware IM User Guide).

Many objects follow a certain life cycle – they go through a sequence of states, each of them having a particular logical meaning. For example, a loan can be in "Current" or "Past" states. For objects with a well-defined life cycle most business rules identify which transitions from one state to the other are allowed and what happens with each transition.

Not all objects need to go through a sequence of states (or even have states for this matter). In our application we have both types of objects – objects with a life cycle (Loan, Reservation) and objects without it (Member, Item, ItemType[3]).

4.2.1 Item Object

We have defined two calculated attributes for the Item object – "Available" and "CurrentReservation" (see "Calculated Attributes" section above), so we need to define two rules that calculate these attributes.

"Item availability" Rule.
An item is available if it is not on loan, if it is not reserved and if it is not withdrawn by the library. Otherwise the item is not available. The "not on loan" condition means that there is no loan in the "Current" state associated with the item. Using the Rule Language this condition can be written as:

```
NOT (EXISTS Loan WHERE (Loan IN Item.Loans AND
Loan.Status='Current'))
```

The condition that there is no current reservation associated with the item, can be written like so:

```
Item.CurrentReservation IS UNDEFINED
```

The entire rule can therefore be written like so:

```
IF
```

[3] Although such objects as Member and Item have states most of the time these objects are in one state ("Active" for Member and "Released" for Item) so effectively we can say that they do not have a life cycle.


```
  NOT (EXISTS Loan WHERE (Loan IN Item.Loans AND
Loan.Status='Current'))
AND
  Item.CurrentReservation IS UNDEFINED
AND
  Item.InternalStatus='Released'
THEN
  Item.Available='Yes'
ELSE
  Item.Available='No'
```

“Current reservation” Rule.

The rule that calculates the current reservation needs to sort all waiting or offered reservations associated with the item by the reservation date and take the one with the most recent date. In the Rule Language this can be written as:

```
FIND Reservation WHERE Reservation IN Item.Reservations AND
(Reservation.Status='Waiting' OR Reservation.Status='Offered')
ORDER BY Reservation.ReservedOn TAKE BEST 1
Item.CurrentReservation=Reservation
```

We should also consider the fact that we should only do this if we do not already have the current reservation that has been offered but not taken up. This condition can be written as:

```
Item.CurrentReservation.Status<>'Offered'
```

The entire rule therefore can be written like this:

```
IF Item.CurrentReservation.Status<>'Offered'
THEN
  FIND Reservation WHERE Reservation IN Item.Reservations AND
(Reservation.Status='Waiting' OR Reservation.Status=’Offered’)
ORDER BY Reservation.ReservedOn TAKE BEST 1
  Item.CurrentReservation=Reservation
```

Note that when writing these rules we do not need to concern ourselves with the circumstances under which these rules should be executed, nor different scenarios that lead to the execution of these rules[4]. We focus entirely on the rules of the object and rely on the system to trigger the rules whenever necessary.

[4] We should keep in mind though that the rules are triggered when the instance of the object is created or modified. Our two rules therefore will always be executed whenever this happens. If we are concerned about performance we could optimise the rules slightly and for the second rule, for example, add a condition that will guarantee that the rule is only executed when there is a change in the list of reservations of the item or a reservation belonging to this list is modified (if the list of reservations does not change the rule will not change anything either). The condition looks like so:

```
IF Item.Reservations WAS CHANGED OR Reservation FROM Item.Reservations WAS
CHANGED
```


Configuration of Rules.

Let us now enter these rules into ***Aware IM***. To do this double click on the node of the Item business object in the Elements Tree and select the "Update Rules" tab at the bottom of the screen. Click on the  icon to create a new rule. Enter the name of the rule "Item availability". A new empty rule will appear in the list. We will see the rule editor underneath the list. The editor consists of 2 tabs – the "Standard Form" tab and the "Textual Form" tab. Both tabs allow entering the text of the rule – the "Standard Form" has a simplified interface but does not allow entering rules in certain formats, whereas the Textual Form allows entering any rules (see also the "Adding/Editing Rules" section in the Aware IM User Guide). In this example we will use the "Textual Form". Click on the "Textual Form" tab and enter the first rule exactly as described above into the text area of the tab, enter the name of the rule "Item availability" and click on the  icon. In a similar fashion enter the second rule (with the name "Current reservation"). Click on the "Save" button in the toolbar (or select File/Save from the menu) to save changes to the rules.

The rules for the Item object have been entered. There are no more rules for this object.

4.2.2 Member Object

For the Member object we have two calculated attributes that we need to enter the rules for – the "MemberNumber" attribute identifying the unique number of a member in the system and the "OutstandingCharges" attribute.

"Membership number" Rule.

The unique number of a member will be calculated when the member is created and for simplicity will be equal to the number of existing members in the system up to this point (which includes the new member). In the Rule Language format this can be written as[5]:

```
IF
  Member IS NEW
THEN
  Member.MemberNumber = MAX Member.MemberNumber+1
```

"Outstanding charges" Rule.

The rule that calculates the outstanding charges will calculate the difference between the amount of fees that the member has been charged and the amount of payments that he has made. In the Rule Language format this rule looks like so:

```
Member.OutstandingCharges=
SUM Fee.Amount WHERE (Fee IN Member.Fees)-
```

[5] In Aware IM 5.1 this can also be achieved without a rule – just by marking the MemberNumber attribute as "auto-incremented"


```
SUM Payment.Amount WHERE (Payment IN Member.Payments)
```

Let us now enter these rules into the configuration. This time we will use the "Standard Form" of the rule editor. Bring up the rule dialog as described in the previous section (4.2.1) and click on the "Standard form" tab. In the first cell of the green area identifying conditions enter "`Member IS NEW`". In the yellow area indicating actions enter "`Member.Number=MAX Member.MemberNumber+1`". Enter the name of the rule "Membership number" and click on the OK button. As you can see the user interface of the Standard Form allows us not to enter "IF" and "Then". It will also spare us from having to enter "`AND`" if there are multiple conditions.

Enter the second rule with the name "Outstanding charges" using either the Standard Form or the "Textual form". There are no more rules for the Member object.

4.2.3 Loan Object

This object will have a lot more rules since it has a distinct life cycle. Let us define this life cycle first before defining the rules.

A loan is created in the "New" state. In this state the loan is considered to be invalid. The loan validates itself and if validation is OK the loan moves itself to the "Current" state. Once the borrowed item has been returned the loan is moved to the "Past" state. It stays in the "Past" state until it is deleted.

Before we start defining the rules we should also consider renewal of the loan. According to the requirements (see 3.2.1) there is a limitation on a number of times a loan can be renewed. The Loan object therefore needs to maintain this number in a separate attribute, which we will call "RenewalCount". We will add this attribute of the "Number" type to the Loan object (initial value of the attribute is 0).

4.2.3.1 Rules of the "New" state

Let us now define the rules applicable to the "New" state of the Loan object. Note that a loan is created in this state (we set the initial value of the Status attribute to "New").

"Item availability validation" Rule.

A loan can be created only for the item that is not on loan already. We obviously check the "Available" flag of the Item object but we also need to consider the situation when the loan is for the reserved item (in this case the item is considered to be unavailable, but the loan converting the reservation must be allowed). The action of the rule must be "`REPORT ERROR`" which indicates that whatever operation the system is executing currently must be aborted as one of the objects in the system is invalid.

We define the following rule with the name "Item availability validation":

```
IF
  Loan.Status='New'
AND
```



```
  Loan.Item.Available='No'
AND
  NOT(Loan.Item.CurrentReservation.Member=Loan.Member AND
Loan.Item.CurrentReservation.Status='Offered')
THEN
  REPORT ERROR 'The item is not available for borrowing.'
```

“Member status validation” Rule.

A loan can be created only for active members:

```
IF
  Loan.Status='New'
AND
  Loan.Member.Status<>'Active'
THEN
  REPORT ERROR 'Non-active members are not allowed to borrow
items.'
```

“Same type items validation” Rule.

A member cannot borrow more than a maximum allowed number of loans for items of a particular type. This can be checked by the following condition:

```
COUNT Loan WHERE (
  Loan IN ThisLoan.Member.Loans
  AND
  Loan.Status='Current'
  AND
  Loan.Item.Type = ThisLoan.Item.Type)
>= Loan.Item.Type.MaxBorrowNumber
```

We count the current loans that belong to our member and have the same type as our item and compare with the maximum value defined in the type of our item. Note that we use “`ThisLoan`” in the condition – this is to differentiate our loan from all other loans that are being searched (see also the “Instance Prefixes” section in the Aware IM User Guide).

The entire rule looks like so:

```
IF
  Loan.Status='New'
  AND
  COUNT Loan WHERE (
    Loan IN ThisLoan.Member.Loans
    AND
    Loan.Status='Current'
    AND
    Loan.Item.Type=ThisLoan.Item.Type)
  >= Loan.Item.Type.MaxBorrowNumber
THEN
```



```
  REPORT ERROR 'Member already has
<<Loan.Item.Type.MaxBorrowNumber>> <<Loan.Item.Type.Name>>s.'
```

"Total items validation" Rule.

A member may not have more than the maximum number of loans. The rule looks like so[6]:

```
IF
  Loan.Status='New'
AND
  COUNT Loan WHERE (Loan IN ThisLoan.Member.Loans AND
Loan.Status='Current') >= 10
THEN
  REPORT ERROR 'Member already has maximum allowed number of
items.'
```

"Due date" Rule.

We must also have a rule that calculates the initial value of the "DueDate" attribute. The rule looks like this:

```
IF
  Loan.Status='New'
THEN
  Loan.DueDate=Loan.LoanDate+Loan.Item.Type.MaxLoanPeriod
```

"Current loan" Rule.

As we said before valid loan should automatically move itself into the "Current" state[7]. The rule looks like this:

```
IF
   Loan.Status = 'New'
THEN
   Loan.Status = 'Current'
```

The rule looks a bit strange. The problem here is that the state transition should only be performed when the loan has been validated. Note that the sequence of all previous rules is not important – we can add them to the Loan object in any order (even the "DueDate" rule – it does not matter if the date is calculated for the invalid loan). In most cases all rules associated with a business object are equivalent as far as their order of execution is concerned – the system picks the order itself at random (see also the "Rule Evaluation" section in the Aware IM User Guide").

[6] At the moment the rule uses fixed number 10 as the maximum number of loans. We will address the requirement that management should be able to change this number from time to time in section 4.5.3.1.

[7] The question here is whether the Loan should move itself into the Current state or the state should be explicitly set to "Current" externally after the loan has been created. There are merits in both approaches – we prefer here to get the loan to set its state automatically and demonstrate rule priorities.

Our automatic transition into the “Current state” is the rare case when the order of execution of a particular rule is important – we want to be sure that the rule that makes a state transition is executed after the validation rules. The mechanism to ensure this is to lower the priority of the “Current state” rule. We can do this when we enter the rule – we need to switch to the “Advanced” tab of the rule editor, click on the “I want to assign priority manually” radio button and specify the lower priority than the default priority of 30 (for example, 20) – see also the “Rule Priorities” section in the Aware IM User Guide.

4.2.3.2 Rules of the “Current” state.

In this state we mostly need to concentrate on loan renewal. We will consider the loan as being renewed when its renewal count is incremented and not zero (we assume that the “RenewalCount” attribute is incremented externally to indicate an attempt to renew the loan).

“Renewed due date” rule.

This is the rule that re-calculates the due date when the loan has been renewed:

```
IF
  Loan.RenewalCount WAS CHANGED
AND
  Loan.RenewalCount > 0
THEN
  INCREASE Loan.DueDate BY Loan.Item.Type.MaxLoanPeriod
```

Note that we can freely perform arithmetic operations with dates by adding a number of days to the due date.

“Renewal limit validation” rule.

A loan cannot be renewed more times than allowed by the type of the item that has been borrowed. This can be captured by the following rule:

```
IF
  Loan.RenewalCount WAS CHANGED
AND
  Loan.RenewalCount > 0
AND
  Loan.RenewalCount > Loan.Item.Type.MaxRenewals
THEN
  REPORT ERROR 'Cannot renew loan, the maximum number of
renewals (<<Loan.Item.Type.MaxRenewals>>) is reached.'
```

“Renewal reservation validation” rule.

A loan cannot be renewed if someone has already reserved the borrowed item while it was on loan. We express this in the following rule:

```
IF
  Loan.RenewalCount WAS CHANGED
AND
```



```
  Loan.Item.CurrentReservation IS DEFINED
AND
  Loan.RenewalCount > 0
THEN
  REPORT ERROR 'Cannot renew loan, there is a reservation for
the item.'
```

“Renewal member validation” rule.
An attempt for loan renewal must be rejected if the library has made a decision to suspend the member who has borrowed the item. Therefore we check that the member status is still “Active”:

```
IF
  Loan.RenewalCount WAS CHANGED
AND
  Loan.Member.Status<>'Active'
AND
  Loan.RenewalCount > 0
THEN
  REPORT ERROR 'Cannot renew loan for non-active member.'
```

“Renewal item validation” rule.
An attempt for loan renewal must be rejected if the library has made a decision to withdraw the borrowed item. Therefore we check that the item status is still “Released”:

```
IF
  Loan.RenewalCount WAS CHANGED
AND
  Loan.Item.InternalStatus<>'Released'
AND
  Loan.RenewalCount > 0
THEN
  REPORT ERROR 'Cannot renew loan, the item is requested back
to library.'
```

4.2.3.3 Rules of the “Past” state

Finally we need to add rules applicable to a loan in the “Past” state.

“Return date” rule.
This rule registers the return date of the loan when it has been closed:

```
IF
  Loan.Status WAS CHANGED TO 'Past'
THEN
  Loan.ReturnDate = CURRENT_DATE
```

“Renewal status validation” rule.
An attempt to renew the loan in the “Past” state must be rejected:


```
IF
  Loan.RenewalCount WAS CHANGED
AND
  Loan.Status<>'Current'
AND
  Loan.RenewalCount > 0
THEN
REPORT ERROR 'Cannot renew non-current loan'
```

These are all the rules of the Loan object. We can now enter these rules into ***Aware IM*** using either the Standard or Textual forms of the rule editor as described in 4.2.1.

4.2.4 Reservation Object

We will now define the rules of the Reservation object. This object also has a distinct life cycle, so let us define the life cycle first.

A reservation is created in the “New” state. Just like the Loan object, a reservation in this state is considered to be invalid. If validation proves to be OK, the reservation automatically moves itself into the “Waiting” state, which indicates that the reservation is waiting for its turn in the queue of reservations and/or is waiting for the item to become available. When the item becomes available and the reservation is first on the queue, the reservation moves to the “Offered” state, which indicates that the reservation has been offered to the member and is waiting for the member to take up the offer. The member can either cancel the reservation, in which case the reservation moves to the “Cancelled” state) or take up the offer by borrowing the reserved item, in which case the reservation moves to the “Converted” state. Reservation is waiting for the member to do something for a limited time. If the member does nothing within this period of time the reservation moves to the “Expired” state. The “Cancelled”, “Converted” and “Expired” states are the final states of the reservation.

Let us now define the rules.

4.2.4.1 Rules of the “New” state.

“Item availability validation” rule.
An item may only be reserved if it has been borrowed by someone. It should not be possible to reserve available items. We capture this in the following rule:

```
IF
  Reservation.Status='New'
AND
  Reservation.Item.Available='Yes'
THEN
  REPORT ERROR 'Item is available for borrowing. No
reservation necessary.'
```


“Item release validation” rule.
An item may only be reserved if the item has been withdrawn while it was on loan:

```
IF
  Reservation.Status='New'
AND
  Reservation.Item.InternalStatus<>'Released'
THEN
  REPORT ERROR 'Item is not released by the library for
borrowing.'
```

“Member status validation” rule.
An item may only be reserved by active members:

```
IF
  Reservation.Status='New'
AND
  Reservation.Member.Status<>'Active'
THEN
  REPORT ERROR 'Non-active members are not allowed to make
reservations.'
```

“Duplicate reservation validation” rule.
A member cannot place multiple reservations for an item. We can check whether the member has already reserved the item by the following condition:

```
EXISTS Reservation WHERE(
  Reservation.Member = ThisReservation.Member
AND
  Reservation.Item = ThisReservation.Item
AND
(Reservation.Status='Waiting' OR Reservation.Status='Offered')
)
```

The entire rule therefore looks like this:

```
IF
  Reservation.Status = ‘New’
AND
EXISTS Reservation WHERE(
  Reservation.Member = ThisReservation.Member
AND
  Reservation.Item = ThisReservation.Item
AND
(Reservation.Status='Waiting' OR Reservation.Status='Offered')
)
THEN
   REPORT ERROR 'The item is already reserved by the member.'
```


“Duplicate loan validation” rule.
It is not allowed to reserve the item, which has already been borrowed by the member. We can check whether the item has been borrowed using the following condition:

```
EXISTS Loan WHERE (
  Loan.Member = Reservation.Member
AND
  Loan.Item = Reservation.Item
AND
  Loan.Status='Current')
```

The entire rule therefore looks like this:

```
IF Reservation.Status = ‘New”
AND
EXISTS Loan WHERE (
  Loan.Member = Reservation.Member
AND
  Loan.Item = Reservation.Item
AND
  Loan.Status='Current')
THEN
  REPORT ERROR 'Cannot reserve item that is on loan to the
member.'
```

“Waiting reservation” rule.
This is the rule that automatically moves the reservation into the “Waiting” state once all validations have been performed:

```
IF
  Reservation.Status = ‘New’
Then
  Reservation.Status = ‘Waiting”
```

The logic behind this rule is the same as the logic behind the “Current loan” rule in the Loan object (see 4.2.3). We will need to make sure that the priority of this rule is lower than priorities of other rules.

4.2.4.2 Rules of the “Waiting” state

“Reservation fee” rule.
When a reservation is registered the member who made the reservation is charged the appropriate fee. This means that when the reservation is moved into the “Waiting” state the Fee object must be created. The fact that the reservation has been moved into the “Waiting” state can be checked by the following condition:

```
Reservation.Status WAS CHANGED TO ‘Waiting’
```


Note that here we are not checking that the reservation state is *equal* to “Waiting”, because in this case the fee would be charged every time there was a change to the reservation object and the object was in the “Waiting” state (which can happen if, for example, some attribute of the Reservation object was changed – this would trigger the execution of rules).

The entire rule looks like so:

```
IF
  Reservation.Status WAS CHANGED TO 'Waiting'
THEN
  CREATE Fee WITH
Fee.Member=Reservation.Member,Fee.Amount=2.00,Fee.Description=
'Reservation fee for <<Reservation.Item.Title>>'
```

“Offered reservation” rule.

This is the rule that moves the reservation from the “Waiting” state into the “Offered” state provided that the reservation is first on the queue and the item became available. We can check that the reservation is first on the queue by comparing it with the “CurrentReservation” in the reserved item:

```
Reservation = Reservation.Item.CurrentReservation
```

We can check that the item has been returned by checking the list of loans of the item and verifying that there are no “Current” loans any more:


```
NOT (
  EXISTS Loan WHERE (
    Loan IN Reservation.Item.Loans
    AND
    Loan.Status='Current'
  )
)
```

The entire rule can therefore be written like this:

```
IF
  Reservation.Status='Waiting'
AND
  Reservation=Reservation.Item.CurrentReservation
AND
  NOT(
    EXISTS Loan WHERE (
      Loan IN Reservation.Item.Loans
      AND
      Loan.Status='Current'
    )
  )
THEN
  Reservation.Status='Offered'
```

Note that we do not have to capture the precise moment when the item has been returned in the rule – the system will ensure that our rule is triggered when this happens, our rule will be executed and the state of the reservation will be changed. After this the rule will not be invoked again, because the state of the reservation will no longer be “Waiting”.

4.2.4.3 Rules of the “Offered” state

“Item offered on” rule
This rule registers the value of the “OfferedOn” attribute:

```
If
  Reservation.Status WAS CHANGED TO 'Offered'
Then
  Reservation.ItemOfferedOn = CURRENT_DATE
```

Here we capture the exact moment when the state of the reservation changes. If we just checked for equality the date would be reset to the new current date every time the reservation object in the “Offered” state changes.

“Converted reservation” rule
This rule moves the reservation into the “Converted” state once the member has taken up the offer. We check that the member has taken up the loan by the following condition:


```
EXISTS Loan WHERE (
  Loan IN Reservation.Item.Loans
  AND
  Loan.Status = 'Current'
  AND
  Loan.Member = Reservation.Member
)
```

The action of the rule should not only change the status but also register the "BorrowedOn" date. The entire rule, therefore, looks like this:

```
IF
  Reservation.Status = 'Offered'
AND
  EXISTS Loan WHERE (
    Loan IN Reservation.Item.Loans
    AND
    Loan.Status = 'Current'
    AND
    Loan.Member = Reservation.Member
  )
THEN
  Reservation.BorrowedOn = CURRENT_DATE
  Reservation.Status = 'Converted'
```

Note that just like with the "Offered reservation" rule we do not have to check the precise moment when the member actually borrowed the item[8].

4.2.4.4 Rules of the "Cancelled" state

When defining the rules applicable to this state we assume that a reservation is moved into the "Cancelled" state externally, that is, not by rules of the object. The rules should just react to the external change of the object state.

"Cancellation validation" rule.

It is only possible to cancel current reservation, so this rule checks that the change to the "Cancelled" state only happens when the object is either in the "Waiting" or "Offered" state. Since the change of the object's state has already taken place we need to check the value of the state before the change. This can be achieved using the `OLD_VALUE` function (see also the "Functions" section of the Aware IM User Guide).

[8] Just for illustration purposes we can show what the condition would look like if we wanted to capture this moment:

```
Loan WAS ADDED TO Reservation.Item.Loans
AND
AddedLoan.Status='Current'
AND
AddedLoan.Member=Reservation.Member
```


The rule looks like this:

```
IF
  Reservation.Status WAS CHANGED TO 'Cancelled'
AND
  OLD_VALUE(Reservation.Status)<>'Waiting'
AND
  OLD_VALUE(Reservation.Status)<>'Offered'
THEN
  REPORT ERROR 'Cannot cancel non-waiting reservation'
```

These are all the rules of the Reservation object. The only state we have not covered is “Expired”. The object is moved to this state externally, not by rules of the object. We will explain how this happens in 4.3.16.1.

We can now enter the rules of the Reservation object into ***Aware IM***.

4.2.5 Payment Object

We will add a validation rule that checks that payment amount does not exceed outstanding charges of a member (“Payment amount validation rule):

```
IF
  Payment.Amount > Payment.Member.OutstandingCharges
Then
  REPORT ERROR 'Payment amount cannot exceed outstanding
charges.'
```

4.3 Operations

In all previous discussions we have not even mentioned the user interface of our system – we focused entirely on business objects, relationships and business rules that evolved naturally from the requirements. The next step in the configuration of the application is to go through the scenarios of user interaction with the system and define the operations that users will perform with the system.

By carefully studying the requirements again we can identify the following operations that a user will need to perform with the system (again we are ignoring some of the requirements for now – we will return to them later):

- Register new member.
- Find member by name (full or partial) and by number (full or partial).
- Change member details.
- Register a new item.
- Find an item by title, author and/or a code (full or partial)
- Change item details.
- Register new item type.

- Change item type details.
- Borrow an item.
- Renew loan.
- Return an item.
- Reserve an item.
- Cancel a reservation.
- Register a fee for a member.
- Register a payment.

4.3.1 "Register new member" operation

Users will invoke this operation from the system menu. The system menu will have the "New" item for creation of different entities of our system. The "Library member" sub-item of the "New" item will create a new member.

The operation should create a new instance of the "Member" business object. So we need to define the operation in the system menu and instruct the system to create a new instance of the Member object.

In the Configuration Tool the system menu is defined in the *visual perspective* (see also the "Visual Perspective" section in the Aware IM User Guide). The system automatically creates the default visual perspective for administrators called "Administrator". Let us open this perspective for editing. To do this expand the "Visual Perspectives" node in the Elements Tree and double click on the "Administrator" node. The editor of visual perspectives will appear in the working area of the screen. Expand the "Top Bar" entry in the list of "Frames". The entry will have one child called "Menu". Expand this entry to see all the menu items of the menu. Now click on the entry called "Menu" and then click on the  icon to add a new menu item. In the popup dialog that appears enter the name of the menu item "New" and leave the type of the item to be "Folder". Now click on the node of the newly created item in the menu pane and click on the  icon again. Specify the name of the new item - "Library member". Click on the Details link next to "Command to execute" and then select the "Create Object" as the command type. Select "Member" from the list of business objects and click OK for the Command Settings dialog and OK for the New Menu Item dialog.

Click on the "Save" button in the toolbar or select File/Save in the menu to save the changes.

This is literally all we have to do to enable this operation in our application. ***Aware IM*** will take care of all other issues – it will automatically generate and display the form for the user to enter the details of the new member once she selects this operation from the menu. Once the user submits the new details it will invoke the rules of the Member object that we have defined at the previous step to validate the object and calculate the values of calculated attributes and it will save all the details in the database.

4.3.2 "Find member" operation

This operation will allow users to find a particular member. The user will select the operation from the system menu and the system will prompt the user to enter the last name and/or the number of the member that she wants to find, and the system will display the details of this member. The user may also enter a partial name and/or number of the member(s) in which case the system will show all members whose names contain the entered partial name and/or whose numbers contain the entered partial number.

For us to configure such an operation we need to define a *query* that will look for members by name and/or number (see also the "Data Retrieval" section in the Aware IM User Guide). To do this right click on the "Queries" node in the Elements Tree and select "New" from the pop-up menu. The query editor will appear in the working area. Enter the name of the new query "Member" in the list of properties of the query. Select "Member" from the drop down as the business object that the query will be looking for (What to Look For section). We will now define conditions of the search. Conditions are defined in the "Where" panel of the editor. To define the first condition:

1. Click on the cell in the first row of the Attribute/Expression column and select "Member.LastName" from the drop down that appears.
2. Click on the cell in the first row of the "Criterion" column and select "Contains" from the drop down.
3. Click on the cell in the first row of the "Value/Expression" column and select "(Ask at run-time)" from the drop down.

To define the second condition:

1. Click on the cell in the second row of the Attribute/Expression column and select "Member.MemberNumber" from the drop down that appears.
2. Click on the cell in the second row of the "Criterion" column and select "Contains" from the drop down.
3. Click on the cell in the second row of the "Value/Expression" column and select "(Ask at run-time)" from the drop down.

We have now defined two conditions that check whether the member's last name contains the text entered by the user at run time and whether the member's number contains the number entered by the user at run time.

After the query has been run the system may find one or more instances of the Member object that match the conditions. The system will display a table listing the found instances. For each instance the system will display values of certain attributes, so that a user can identify which instances are shown. We can define which attributes will be shown. Click on the "Display As" property in the list of properties and tick "FirstName", "LastName", "MemberNumber", "DateOfBirth" and "EmailAddress" in the list of "Attributes to Display". Then click OK.

We will also sort the results by member's last name. To specify the sorting criteria click on the cell in the "Attribute" column of the "Sorting of results" table and select "LastName". Now click on the cell in the "Order" column and select "Ascending".

Our query has been completed – we can click on the “Save” button and the new query will appear in the Elements Tree.

Now we can define the menu item that will run the query that we have defined. We do it in a similar fashion to how we defined the “New library member” menu item (see 4.3.1). In the “Administrator” visual perspective we define the “Find” folder item and underneath it we define the “Library member” item of the “Run Query” type. We select the “Member” query that we have just defined as the query that will be run.

4.3.3 “Change member details” operation

To change details of a particular member users would need to find the member first and then when the system displays the found instance users would click on the “Edit” button next to the member entry to bring up the member form which they can edit and save.

We have already defined the operation that will find a particular member(s). We need to add the operation that will bring up the form of the member once the system displays a list of members that match the results of the search. To define such an operation we need to edit the “Member” query. Double click on the node representing the “Member” query in the Elements Tree. In the query editor click on the “Operations with records” property in the list of query properties to display a dialog and then click on the “Add” button. In the dialog that appears enter the name of the operation “Edit”, select the type of the operation “Edit” and click on the OK button. The new operation appears in the list of operations. Once we are here we will also define the “Delete” operation so that the user will be able to delete the selected member. We click on the “Add” button again, define the operation with the name “Delete” of the “Delete Object”.

We click on the OK button to close the dialog and then on the “Save” button to save the “Member” query.

4.3.4 “Register new item” operation

This operation is very similar to the “Register new member” operation (see 4.3.1). In the “Administrator” visual perspective we define a new menu item under the “New” folder called “Item”. The type of this menu item should also be “Create Object” and we should select the “Item” object as the object to be created.

4.3.5 “Find item” operation

This operation is similar to the “Find member” operation (see 4.3.2). We could define the “Item” query that would find an item by author, title and code. In this case, however, we will give different categories of users different abilities to perform a search on items. Operators will have the ability to search by *any* attribute of the “Item” object using the form of the “Item” object. If an operator enters any information in any field on the form the system will perform a search based on the entered

information only, ignoring any blank fields. For more details on how this works see the “Searching for Data using Forms” section in the “Aware IM User Guide”.

To give operators the ability to perform the form-based search we will configure a query called “Item using form”. When adding this query we will click on the “Use form” property in the list of query properties and then select the form of the Item object to use. This will indicate that the query will not have any conditions – all conditions will be determined at run time when a user fills in the query form.

We will also define the “Item” query that will search items by author, title and code – we will use this query for members only (we will cover this later). We can also allow operators to use this method as an alternative to a form-based search. Configuring such a query is very similar to configuring the “Member” query.

4.3.6 “Change item details” operation

Configuring this operation is very similar to the “Change member details” operation (see 4.3.3) – we add the “Edit” operation to the “Item” query.

4.3.7 “Register new item type” operation

This operation is very similar to the “Register new member” operation (see 4.3.1). In the “Administrator” visual perspective we define a new menu item under the “New” folder called “Item type”. The type of this menu item should also be “Create Object” and we should select the “ItemType” object as the object that will be created.

4.3.8 “Change item type details” operation

To change the details of the ItemType object users would need to locate the item type first. They can do it either from the form of a particular item or by running a query that finds item types. As normally there will be very few item types in the system we will define the “All item types” query that will find all item types. To define such a query we create a new query as described in the “Find Item” operation (see 4.3.5), select “ItemType” as the object that the query will be looking for, but we do not define any conditions.

We select “Name”, “MaxBorrowNumber”, “MaxLoanPeriod” and “MaxRenewals” as the attributes that will be displayed and we sort the results by “Name” in ascending order. We also add “Edit” and “Delete” item operations to the query.

Once the query has been defined we add a new item in the visual perspective under the “Find” item called “Item types” of the “Run Query” type that runs the newly defined “All item types” query.

4.3.9 “Borrow item” operation

For the “Borrow Item” operation users would need to specify the item that is to be borrowed and the member who borrows the item. One possibility is to invoke this operation from the system menu so that when users select this operation the system

then asks to identify the member and then the item (or the other way around). However, we prefer not to have this operation in the system menu, because most of the time users will already have items or members on their screens as a result of some other operation with the system. So the “Borrow Item” operation will be performed when either some item or some member has been already found by the system (in ***Aware IM*** terminology an item or a member is said to be in the *context*). For example, a user might have run a query that found some items and the user is looking at the screen that lists those items or the user might be working with the form of a particular item or member. The “Borrow item” operation will thus be invoked from the following screens:

1. When a member is being edited and a user sees the form of the member
2. When an item is being viewed/edited and a user sees the form of the item
3. When the “Item” query has been run and a user sees a list with one or more items

Let us look at each of these scenarios:

Invoking the “Borrow Item” operation from the Member form

In this scenario users need to click some button starting the “Borrow Item” operation while they are working with the form of a member. The system already knows the member who wants to borrow (this is the member being viewed/edited) but it needs to ask the user to identify the item to be borrowed. Once the user identifies the item the system needs to create an instance of the Loan object. In fact, this is all the system has to do, since after the Loan has been created, the business rules of the Loan will take over – they will validate the loan and move it to the “Current” state if it is valid.

So the system needs to perform two steps – ask the user to identify an item and create an instance of the Loan object. These steps can only be performed by rule actions, so we have to create a *process* (see also the “Processes as Links between the Business Logic and the User Interface” and “Adding/Editing Processes” section in the Aware IM User Guide). A process usually defines a sequence of actions that are executed one after the other. The actions we need for the “Borrow Item” operation are these:

```
PICK FROM Item
CREATE Loan WITH Loan.Item = Item, Loan.Member = Member
```

The first action will run the “Item” query (that we have already defined – see 4.3.5). Then it will display the results of the query to the user prompting her to pick one item from those displayed. The second action will create the Loan object and initialize its reference attributes with the item that the user has picked at the previous step and the member instance (which is assumed to be in the context of the process). As we said before everything else will be handled by the business rules of the Loan object that are executed when the loan is created.

Let us now define the process in the Configuration Tool. Right click on the “Processes” node in the Elements Tree and select “New” from the pop-up menu. The

process editor appears in the working area of the screen. Enter the name of the new process “BorrowItem” in the list of process properties. Our process expects that some instance of the Member object is already in the context when the process starts (this is the member being viewed/edited), so we must specify “Member” as *process input*. We click on the Input property and select the Member object from the list of available objects and click on the right arrow button to move the Member object to the list of objects selected as process input. We can now define the actions of the process. We click on the “Textual rules” tab located at the bottom of the editor and enter the above actions using the familiar rule editor (see 4.2.1). We can now click on the “Save” button to save changes. The new process will appear in the Elements Tree.

Now we need to define the operation that will start the “BorrowItem” process from the form of the Member object. To do this open the Member object for editing and select the “Main” form in the “Forms” tab of the object editor. The “Main” form is the form that ***Aware IM*** creates automatically when a new object is defined (see also the “Business Object Forms” section in the Aware IM User Guide). Click on the “Panel Operations” property in the list of properties of the form. Then select the Add Operation button. In the operation dialog specify the name of the operation “New Loan”, select its type (Start Process) and – select the “BorrowItem” process from the list of processes. Then click OK and save the changes in the Member object. At runtime ***Aware IM*** will automatically create a button for the user to click on to invoke the process.

Note that ***Aware IM*** will automatically put the instance of the Member object being viewed/edited into the context and will use it as process input when the operation is invoked.

Invoking the “Borrow Item” operation from the Item form

This scenario is quite similar to the previous one. In this scenario users need to click a button starting the “Borrow Item” operation while they are working with the form of an item. The system already knows the item that is to be borrowed (this is the item being viewed/edited) but it needs to ask the user to identify the member who borrows the item. Once the user identifies the member the system needs to create an instance of the Loan object.

Just like in the previous scenario we create a process (called “BorrowItem2”) with the Item object as its input and the following actions:

```
PICK FROM Member
CREATE Loan WITH Loan.Item = Item, Loan.Member = Member
```

We configure this process in a very similar fashion to the “BorrowItem” process. Then we define an operation that starts the BorrowItem2 process from the form of the Item object in much the same way we did it for the form of the Member object.

Invoking the "Borrow Item" operation from query results

In this scenario we assume that a user has run the "Item" query and is looking at the screen that shows one or more items. We want the user to be able to click on the "Borrow" item button located next to the item if the user wants to create a loan for this particular item. The system is then expected to ask the user to identify the member that borrows the item and create the corresponding loan.

We already have the process ("BorrowItem2") that performs the required action – we just need to invoke it from the query results screen. In order to do this we open the "Item" query for editing and define the new operation called "Borrow" of the "Start Process" type. We select the "BorrowItem2" process as the process to be started. Then we save the changes to the query.

Note that ***Aware IM*** will automatically put the instance of the Item object, on which the user clicked when she invoked the operation, into the context and will use it as process input.

4.3.10 "Renew Loan" operation

To renew a loan, users would need to navigate to the loan that is to be renewed and then click on the "Renew" button. Users can navigate to the loan either from the form of a member that borrowed the item (the form of the member will list the loans of the member because the Member object has the "Loans" reference attribute shown on the form) or from the form of the item (which also has a list of loans for the item).

To renew a loan the system would need to increment the "RenewalCount" attribute of the Loan object and the rest will be taken care by the business rules of the Loan object (see 4.2.3). In order to do this we would need to define the process called "RenewLoan" that requires the Loan object as its input and which has the single action:

```
INCREASE Loan.RenewalCount BY 1
```

In fact, we will also add another action to the process that will report to the user the date that the loan has been extended to[9]:

```
DISPLAY MESSAGE 'Loan has been extended until
<<Loan.DueDate>>'
```

We configure this process in the same fashion as we configured the "BorrowItem" process (see 4.3.9). ***Aware IM*** will also automatically define the operation to invoke the process from the form of the Loan object.

[9] If renewal attempt fails the corresponding validation rules of the Loan object would issue the "REPORT ERROR" action which would abort the execution of the process and the system would never execute the DISPLAY MESSAGE action

4.3.11 "Return Item" operation

To return an item a user would need to navigate to the corresponding loan and then click on the "Close" button. A user can navigate to the loan either from the form of a member that borrowed the item or from the form of the item (see 4.3.10).

To close a loan the system would need to change the status of the loan to "Past" and the rest will be taken care by the business rules of the Loan object (see 4.2.3). In order to do this we would need to define the process called "CloseLoan" that requires the Loan object as its input and which has the single action:

```
Loan.Status = 'Past'
```

We configure this process in the same fashion as we configured the "BorrowItem" process (see 4.3.9). ***Aware IM*** will also automatically define the operation to invoke the process from the form of the Loan object.

4.3.12 "Reserve Item" operation

This operation is very similar to the "Borrow Item" operation (see 4.3.9). The "Reserve Item" operation will be invoked from the following screens:

1. When a member is being edited and a user sees the form of the member
2. When an item is being viewed/edited and a user sees the form of the item
3. When the "Item" query has been run and a user sees a list with one or more items

Just like with the "BorrowItem" operation we will need to define two processes – "ReserveItem" and "ReserveItem2". The "ReserveItem" process will require the Member object as process input; it will run the query to find an item and then will create the Reservation object (properly initialized):

```
PICK FROM Item
CREATE Reservation WITH Reservation.Item=Item,
Reservation.Member=Member
```

The "ReserveItem2" process will require the Item object as process input; it will run the query to find a member and then will also create the Reservation object:

```
PICK FROM Member
CREATE Reservation WITH Reservation.Item=Item,
Reservation.Member=Member
```

Aware IM will automatically define operations that will invoke the "ReserveItem" process from the Member form and the "ReserveItem2" process from the Item form. We will also define the "Reserve" operation for the "Item" query to handle scenario 3 above.

4.3.13 "Cancel Reservation" operation

To cancel a reservation users would need to navigate to the corresponding reservation and then click on the "Cancel" button. Users can navigate to the reservation either from the form of a member who made the reservation or from the form of the item (this is similar to the "Renew Loan" operation – see 4.3.10).

To cancel a reservation the system would need to change the status of the reservation to "Cancelled" and the rest will be taken care by the business rules of the Reservation object (see 4.2.4). In order to do this we would need to define the process called "CancelReservation" that requires the Reservation object as its input and which has the single action defined:

```
Reservation.Status = 'Cancelled'
```

We will also ask the user to confirm cancellation of the Reservation and check that he replied "Yes" before changing status of the reservation. The whole process will look like so:

```
DISPLAY QUESTION 'Do you want to cancel the reservation?'
If Question.Reply='Yes' Then
  Reservation.Status='Cancelled'
```

4.3.14 "Register Fee" operation

Just like with "Borrow Item" (4.3.9) or "Reserve Item" (4.3.12) operations users will invoke the "Register Fee" operation from the form of a member. The member being viewed/edited identifies the member who is being charged.

When the user invokes the operation the system will let the user fill in the details of the fee – such as amount and description. Essentially the system will need to create the instance of the Fee object. In this particular case, however, we can't just define the menu item of the "Create Object" type because the system will also need to initialize the Fee object with the reference to the member. This can only be done by the rule action, so we need to define the "Charge Fee" process, which requires the Member object as its input and contains the following action:

```
ENTER NEW Fee WITH Fee.Member = Member
```

This action will display the form of the Fee object to the user (already initialized with the reference to the appropriate member). When the user submits the form the system will create the instance of the Fee object.

Once we have defined this process ***Aware IM*** will have automatically created the operation that can be invoked from the form of the Member object.

Unlike other operations involving a member we prefer this one to be in the system menu as well. When this operation is invoked from the system menu there is no

"member" context and so the system needs to ask the user to identify the member that will be charged. Therefore we need to define another process called "ChargeFee2" that does not require any input and has the following actions:

```
PICK FROM Member
ChargeFee
```

The first action will display the results of the "Member" query (see 4.3.2). The second action calls the "ChargeFee" process that we have already defined. Note that ***Aware IM*** will automatically use the member that the user picks at the previous step as input for the "ChargeFee" process[10].

We can now define the "Member" folder menu item in the visual perspective and underneath it define the new menu item called "Charge fee" of the "Start process" type that starts the "ChargeFee2" process.

4.3.15 "Register Payment" operation

This operation is very similar to the "Register Fee" operation (see 4.3.14). Users will also invoke this operation both from the system menu and from the form of the Member object. We will also need to define two processes – the "RegisterPayment" process and the "RegisterPayment2" process.

The "RegisterPayment" process requires the Member object as its input and has the following action:

```
ENTER NEW Payment WITH Payment.Member = Member
```

The "RegisterPayment2" process does not require any input and has the following actions:

```
PICK FROM Item
RegisterPayment
```

4.3.16 Expiring Reservations and Processing Overdue Loans

We have now basically defined all the main operations that a user will perform in our application. One topic we have not covered yet is how reservations expire and how overdue loans are processed.

4.3.16.1 Expiring Reservations

A reservation for an item should expire if a member who had reserved the item failed to borrow the item within a certain time frame after the item became available (see requirements in 3.2.4). The system, therefore, needs to check "offered" reservations

[10] Note that instead of calling the ChargeFee process we could just repeat the same action that the ChargeFee process executes as we did in the previous processes. Here, however, we want to demonstrate the action that calls other processes.

regularly to see if any of them have expired and if so, change their status to “Expired” (see 4.2.4).

The feature of the Configuration Tool that allows checking something at regular time intervals is called “Scheduling” (see also the “Scheduling” section in the Aware IM User Guide). Basically you define rules that specify how often checks are being made. The check itself has to be defined as a process. In our case we need to define a process that will check the “offered” reservations and change their status if necessary. This process will be regularly called by the scheduling rules.

So we start by defining the “ExpireReservations” process that does not require any input and has the following actions:

```
FIND Reservation WHERE
  Reservation.Status='Offered'
  AND
  DAY_DIFFERENCE(Reservation.ItemOfferedOn,CURRENT_DATE)>6
Reservation.Status = 'Expired'
```

The first action finds all reservations in the “Offered” state that have been offered more than 6 days before the day when the check is being made. The second action changes the status of the reservation to “Expired”. Note that ***Aware IM*** will automatically change the state of *all* reservations found by the previous action. If no reservations were found ***Aware IM*** will do nothing.

Now that we have defined the process we need to define when this process is going to be called. We right click on the “Scheduling” node in the Elements Tree and select “New” from the pop-up menu. We then click on the  icon to add a new scheduling rule. We add the name of the rule and the rule entry appears in the list of rules. We can now use the Standard View to add the rule itself. We select the “Daily” radio button and specify the time of the day when the system will start the process, for example 01:00. Then we select the “ExpireReservations” as the process that the system will run. We can now click on the “Save” button to save the changes.

4.3.16.2 Processing Overdue Loans

Our application should be regularly checking if a loan is overdue and if so, charge the appropriate fee and send the reminder e-mail to the member. This means that we need a scheduling process to do this. We also need to store the last time the overdue fee was charged so we can check whether we need to charge another fee. Therefore, we need to do the following:

1. Define a new calculated attribute of the Loan object called “OverdueFeeChargedOn” of the “Date” type
2. Add “Overdue fee” rule to the Loan object that will react to the change of the “OverdueFeeChargedOn” attribute:

```
IF
  Loan.OverdueFeeChargedOn WAS CHANGED
```



```
THEN
CREATE Fee WITH
  Fee.Member=Loan.Member,
  Fee.Amount=1.00,
  Fee.Description='Loan overdue fee for '+Loan.Item.Title
```

3. Define the process “ProcessOverdueLoans” with the following actions (note that we will cover reminder e-mails and letters later – see 4.5.1).

```
FIND Loan WHERE
  Loan.Status='Current'
  AND
  Loan.DueDate<CURRENT_DATE
  AND
  (Loan.OverdueFeeChargedOn IS UNDEFINED
   OR
   DAY_DIFFERENCE(Loan.OverdueFeeChargedOn, CURRENT_DATE) > 6)

Loan.OverdueFeeChargedOn=CURRENT_DATE
```

4. Define a new scheduling rule that invokes the “ProcessOverdueLoans” process on a daily basis at 03:00

4.3.17 Testing the Application

At this stage we basically have the core of the application ready. There is still a fair amount of work to be done – polish up the user interface, add reports and documents, define access control rules. It is highly recommended, though, that in the process of application configuration these issues be tackled later. First we need to make sure that the core of the application is working fine before worrying about the more nitty-gritty details.

So the next step is to test the application configured so far – we can check how the operations work, whether rules are executed as expected etc. Before we can start testing the application we need to put the business space version under test (see also the “Testing Mode” section in the Aware IM User Guide). To do this right click on the business space version node in the Elements Tree and select “Put under Test” from the pop-up menu. At this point ***Aware IM*** will build all the required database tables and links between the tables. Once the version has been put under test, its state will be changed from “New” to “Under Test” and it will be displayed in pink.

We now need to start using the system pretending to be the system’s operator. First of all we need to start the Internet browser and log into the system. We can do this by selecting “Tools/Browser Login” command in the menu of the Configuration Tool. ***Aware IM*** will start the default browser and display the login screen. We need to type in the name of our business space, “admin” as user name, “password” as password and tick the “Testing mode” checkbox. When we click on the “Logon” button ***Aware IM*** will start the application we have configured.

We can now create a few items and members, borrow and reserve items, search for members and items and perform all other operations that we have defined. Note that all objects that we create during testing will be stored in a separate database, which means that if we had any real operation data it would not be affected by the testing data.

4.4 Fine-tuning the Application

Now that we have tested the core of our application and made sure that everything works as expected we can start fine-tuning the application by adding some missing functionality, improving user interface and paying more attention at how the information is presented to the user.

To continue working with the business space version after it has been put under test we need to start editing it. Right click on the business space version node in the Elements Tree and select “Start editing” from the popup menu.

4.4.1 Documents

We will start our application improvements by adding a couple of documents to the configuration.

4.4.1.1 Printable statement

With this feature users will be able to invoke the operation that will display a PDF document listing loans, reservations, fees and payments of a particular member. Users will then be able to print out this document. Users should be able to invoke this operation while they are viewing/editing the form of a particular member.

Before we can configure such an operation we need to configure a document. This document will need to list loans, reservations, fees and payments of a member. There are different types of documents that ***Aware IM*** supports (MS-Word, MS Excel, Text etc), however, the only document type capable of iteration is “Report” (see also the “Reports” section in the Aware IM User Guide). Also, report is only capable of displaying lists of one particular object and we need to display lists of several different objects. Fortunately report supports a concept of “sub-reports”, where there is a single master document containing one or more sub-report documents, each of them performing its own iteration. This means that we need to define several reports – one to display loans, another - to display reservations etc; and we also need a master report to combine all these reports in one document.

We will start by configuring documents that will be used as sub-reports within our master document. To add a new document to the configuration right click on the “Document templates, reports” node in the Elements Tree and select “New” from the pop-up menu. Enter the name of the new document “Member payments” in the list of document properties and select the type of the document “Report”. The report editor will be displayed in the working area of the screen. The Report Designer is described in detail in the “Working with Report/Presentation Designer” section of the Aware IM User Guide and we will not be going through it in detail here. You can have a look at

the final document in the sample configuration of the Library application located in the "samples" directory of your Aware IM installation.

Once we are happy with the layout of the document we click on the "Save" button in the toolbar or select File/Save from the menu of the Configuration Tool. The new document appears in the list of documents in the Elements Tree.

In a similar fashion we can define other documents that will be used as sub-reports – "Member loans" (to display Loan objects), "Member fees" (to display Fee objects) and "Member reservations" (to display Reservation objects).

Now we can define the master report called "Member details". The layout of this report includes several sub-report elements (take a look at the final layout of the document in the sample configuration). Note that this report assumes that there is a particular instance of the Member object in the context – that's why it refers to the attributes of the Member object inside its tags. It is also worth drawing your attention to the queries that identify the source of data for sub-report elements. If we look, for example, at the sub-report element that prints out "Member payment" report we will see that the query for the sub-report element looks like this:

```
FIND Payment WHERE Payment IN Member.Payments ORDER BY
Payment.AppliedOn
```

This query finds all payments of the member object that is assumed to be in the context.

Note also that the report has no "Details" section – this is because the report itself does not display any lists (lists are shown inside the sub-reports).

Now that we have finally defined all our documents we can define the operation that will display the master report. The operation will be invoked from the form of the Member object, so we open the Member object for editing, select the "Main" form in the Forms tab, click on the Panel Operations property of the form and then click on the "Add" button to define a new panel operation. We define an operation called "Printable version" of the "Create Document" type and select the name of the document "Member details". Then we save the changes to the Member object. Note that ***Aware IM*** automatically puts the Member object being viewed/edited into the context, so it is available for the "Member details" report, which expects it.

4.4.1.2 Payment receipt

Another improvement that we will introduce is the improvement to the "Register Payment" operation. Once the user finishes registering the payment the system will ask the user if she wants to display the receipt. If the user answers yes, the system will display the page, which lists the details of the member, the payment amount, description etc.

The page that will be displayed if the user asks for the receipt will be represented by an HTML document. We add the new document to our configuration in much the

same way we have defined our reports. The difference is that the type of the document is “HTML” and that we prepare the document template not in the Report Designer but in any HTML editor of our choice. We define a new document of the HTML type, enter its name “Payment receipt” and click on the “Import” button on the documents editor to import the layout of the page that we have prepared in the HTML editor. We need to put all the layout files (including the HTML file and any images and/or styles) into a separate directory and specify the name of this directory when we import the layout. You can have a look at the layout of the “Payment receipt” document in the configuration of the Library application located in the “samples” directory of your Aware IM installation. Note that the document expects the Payment object to be in the context of any operation that is using the document.

Now we need to modify our RegisterPayment process (see 4.3.15) to:
1. Ask the user if she wants the system to display a receipt
2. Display the “Payment receipt” document if required.

We do this by adding the following rules to the RegisterPayment process:

```
DISPLAY QUESTION 'Display receipt?'
IF Question.Reply = 'Yes' THEN DISPLAY DOCUMENT 'Payment
receipt'
```

The entire process, therefore, looks like this:

```
ENTER NEW Payment WITH Payment.Member = Member
DISPLAY QUESTION 'Display receipt?'
IF Question.Reply = 'Yes' THEN DISPLAY DOCUMENT 'Payment
receipt'
```

Note that the first action puts the Payment object that the “Payment receipt” document expects in the context.

4.4.2 Access Control

Now we can address the requirements that our application should be accessed online by members and by guests who need to register themselves with the library system so that they can access it as members (see 3.2.9).

Naturally, members and guests who will be using our application online will have limited access to the application features. Therefore, we need to define access restrictions in our configuration.

4.4.2.1 Member access

Let us think through the scenarios of a member interaction with the system. A member should be able to do the following:

- Log into the system using her own credentials
- View/change her own details, but not details of other members

- Search for library items, but not change details of these items
- Reserve an item or cancel reservation
- Renew her loan
- Make credit card payment (we will look at this one in 4.5.2)

All other operations should be disallowed for a member.

We will look at each of these operations. Before we do this we need to configure the *access level* for members working with the application (see also the "Access Control" section in the Aware IM User Manual). ***Aware IM*** automatically creates the "Administrator" access level, which allows all operations by default, and the "Guest" access level, which disallows all operations by default. You can also define your own access levels. We will define the "Member" access level.

To do this right click on the "Access Levels" node in the Elements Tree and select "New" from the pop-up menu. The editor of the access levels appears in the working area of the screen. Enter the name of the access level "Member" in the list of properties and click on the "Save" button. The new access level appears in the Elements Tree. We will edit this access level as we go through the operations of a member.

Member login

Instances of the Member object will be created either by the operator (see 4.3.1) or by guests when they register themselves with the system (see 3.2.9). In any event when these instances are created the operator or guest should specify the user name and password of the new member (among other details), which means that the Member object should have these attributes defined. As a matter of fact, we do not need to define these attributes – ***Aware IM*** will do it automatically provided that we make the Member object part of the "SystemUsers" group (this is the special group that ***Aware IM*** automatically creates that contains all objects that can access the system). To do this we double click on the "SystemUsers" node (located under the "Business Object Groups" node) . In the editor of groups we select "Member" from the list of available objects and click on the right arrow button to make this object member of the group. We click on the "Save" button to save the changes to the group.

If we now look at the Member object we will notice that ***Aware IM*** has added a few attributes automatically – for example, "LoginName", "Password" and "AccessLevel". The last attribute stores the value of the access level that will be assigned to a member once she logs into the system. We need to assign an initial value "Member" to this attribute (this is the name of our newly created access level) and not allow anyone to change it. Therefore, we open the Member object for editing and set the initial value of "AccessLevel" attribute to "Member".

Viewing/changing own details

We will define access level restrictions that will allow viewing/changing member's own details but not details of other members in the "Member" access level. Let us open this access level for editing. In the editor pane click on the "Business object"

node to expand it and show the list of all business objects. Click on the cell located at the intersection of “Member” row and “Access” column. Select the “Creator:modify only” from the drop down list. We have thus defined the access level for the Member business object that will make sure that a member will only be able to see/change her own details but not details of other members (she will also not be able to create instances of the Member object).

We can now expand the Member object and define access to its attributes. We need a member to be able to view and change the following attributes – “FirstName”, “LastName”, “Address”, “EmailAddress”, “DateOfBirth”, “LoginName” and “Password”, so we leave the access level for these attributes as “Full access”. A member should be able to view but not change the following attributes “MemberNumber”, “Loans”, “Fees”, “Payments” and “Reservation” so we change the access level for these attributes to “Read-only”. Finally a member should not be able even to see the following attributes “AccessLevel” and “Status” so we change the access level for these attributes to “Not available”.

Note that ***Aware IM*** will automatically remove “not available” attributes from the form of the Member object and it will also grey out “read-only” attributes.

Library items

A member must be able to search the library for items and view some details of the items but she should not be able to change any of the details. We can achieve this by setting “read-only” access to the Item object. We open the “Member” access level for editing and change access to the Item object to “read-only” just as we set access to the Member object at the previous step. We also want to make some attributes of the Item object unavailable to member (namely, “Loans”, “Reservations”, “InternalStatus” and “CurrentReservation”), so we set “Not available” access to these attributes.

Reserving items.

A member should be able to reserve items when she is either looking at the results of the query that found some items or when she is viewing the details of a particular item. The operation that reserves items starts the “ReserveItem2” process (see 4.3.12). This process runs the “Member” query to identify a member that reserves the item. When the application is used by a particular member, however, we do not need to run this query, as the member is already known (this is the member using the application). So we modify our process to consider this fact. The process rules now look like this:

```
IF
  LoggedInSystemUser.AccessLevel='Member'
THEN
  CREATE Reservation WITH
Reservation.Item=Item,Reservation.Member=LoggedInMember
ELSE
   PICK FROM Member
   CREATE Reservation WITH
Reservation.Item=Item,Reservation.Member=Member
```


As you can see if currently logged in user has “Member” access level we create the reservation straight away without running the “Member” query. “LoggedIn” prefix in front of the object name can be used to identify the current user of the application (see also the “Instance prefixes” section in Aware IM User Guide).

Other operations.

For all other operations we just need to carefully go through all objects, attributes, processes, queries and documents in the “Member” access level and specify the appropriate access restrictions. For example, as far as processes are concerned we need to allow access to the following processes: “ReserveItem”, “ReserveItem2”, “CancelReservation” and “RenewLoan”. All other processes should be made unavailable.

You can check all access restrictions of the “Member” access level in the sample configuration of the application located in the “samples” directory of the Aware IM installation.

Visual perspective for Members

We now need to define a menu that will be specific to members only. The menu will have only one level and contain the following operations: “My details”, “Search library”, “Home page”.

The type of the “My details” operation should be “Change login details”. The operation of this type will automatically show the form of the current user and allow editing the details. The type of the “Search library” operation is “Run Query” that runs the “Item” query. “Home page” operation is of the “Home page” type – it shows the starting page of the application.

Now we need to create a new visual perspective called “Member” and define the menu as described above for this visual perspective (see 4.3.1). To make sure that this visual perspective is used by the system when members log in, we click on the “Use in login” property of the visual perspective and then tick the “Aware IM will use this perspective when user logs in” checkbox on the dialog that comes up. We will also click on the “Access Levels” button and specify the “Member” access level on this dialog.

To make our application look nicer for members we can create some flashy starting page of the application that will display some information about the library, run some advertisements etc. We can prepare this page in any HTML editor and import the results into the visual perspective for members. To do this expand the “Main” frame, the Main tab and click on the Main content panel under the Main tab. Then select the Contents property in the list of properties of the content panel and select the “Display HTML page“ radio button. Then click on the “Import” button and specify the name of the directory where your custom page (and all its images and styles) are.

4.4.2.2 Guest access

The operations guests can perform in the library application are these:

- Search library for items and check item details (but not change these details)
- Register themselves as library members and specify their details and credentials
- Log into the library system if they are already registered

The task of configuring our system for guests boils down to specifying access restrictions in the “Guest” access level and preparing a visual perspective for guests.

Access level for guests.
Most of business objects, attributes, processes, queries and documents should be unavailable for quests. The only exception is the “Item” query that should be available and the “Item” and “Member” business objects. Access restrictions to the Item object should be exactly the same as in the access level for members. Access restrictions to the Member object should allow access to the following attributes: “FirstName”, “LastName”, ”Address”, ”Gender”, ”EmailAddress”, “LoginName” and “Password”. Access to other attributes should be disallowed. Because guests will be allowed to self-register as members we should assign “Creator:full access” level to the Member object, which guarantees that guests will be able to view/change their own details, not details of other people.

Visual perspective for guests.
The menu for guests will include the following operations:

- Search library (type - “Run Query”, runs the “Item” query)
- Register (type “Register User” for the object “Member”. The operation of this type will automatically bring up the form of the Member object and allow creating the instance of this object)
- Login (type “Login”)
- Home (type “Home page”)

We can add the visual perspective for guests in the same way we have added visual perspective for members (see 4.4.2.1). We should specify that the new visual perspective should be used for the “Guest” access level, import some custom home page and select a particular color style.

4.4.3 Improving forms

Now we can make some improvements to the user interface of the system. We will start with forms of the business objects that the system shows in various parts of the application.

After we have defined our objects (see 4.1) ***Aware IM*** automatically created a form for each object to display its attributes. By default ***Aware IM*** puts all defined attributes on a form in a vertical fashion – one after the other. For many objects this default approach is acceptable. For other objects some improvements may be necessary. Most examples in this section will be performed with the forms of the Member object. Improvements to other objects’ forms will be similar.

4.4.3.1 Creating different forms for entry and editing.

For Member object we prefer to have different forms for editing and entry of the object. Indeed, some of the attributes that the object has are only applicable when the existing object is being viewed/edited and are not applicable when the object is being created. For example, lists of loans, reservations, fees or payments are not applicable when the object is being created so there is no point showing them on the form during creation. So we want to have a separate form, which will be used when the object is being created and which will only have the applicable attributes.

To do this we open the Member object for editing and the first thing we do is make some changes to the "Main" form of the object created automatically. We select the form on the "Forms" tab and we change the name of the form to "Editing" in the list of form properties. Then we un-tick the "Object creation" checkbox under the "Use automatic" category in the property list to indicate that the form will not be used by the system when the object is being created. We click on the Save button to save the changes to the form and click on the  icon to create a new form.

In the dialog that appears we enter the name of the new form "Entry and member registration" (this form will be also be used when guests register themselves as library members – see 4.4.2.2). In the list of properties of the new form we tick the "Object creation" and "User registration" checkboxes. Then we expand the form and double click on the Main form section underneath it to design it. The "Form Section" designer window appears. In this window we can specify which attributes will be shown on the form and in which order. We delete unnecessary attributes and only leave the following attributes "FirstName", "LastName", "Gender", "DateOfBirth", "Address", "EmailAddress", "LoginName", "Password". We use arrow buttons to arrange the remaining attributes in the above order. Then we click on the "Save" button to save changes to the form of the Member object. Now the system will automatically use the newly created form when the Member object is being created.

We can also perform similar changes to the Item object only leaving "Author", "Title", "Description", "Code", "Type", "Location" and "Picture" attributes on the "Entry" form.

4.4.3.2 Creating multi-tabbed form.

Some objects may have large number of attributes and showing them all on one form may be impractical. It may look much better if attributes were split up into several logical groups shown as different tabs. While the Member object is not overly large we still prefer to split its attributes into the following logical groups:

- Attributes that show personal details of the member (name, address etc)
- Attributes showing current activity of the member (her current loans and reservations)
- Financial transactions of the member – fees and payments
- Loan history – all loans of the member

Each of these groups will be shown in its own tab on the "Editing" form of the Member object. In order to split the form in several tabs we need to define a *form section* for each tab (see also the "Form Sections" section in the Aware IM User

Guide). To do this we open the Member object for editing and select the “Editing” form on the “Forms” tab. ***Aware IM*** automatically creates a single form section for the form called “Main” where it puts all the attributes of the object by default. First of all we will make some changes to this default form section. We expand the node of the form and select the “Main” form section underneath it. In the list of properties of the form section we change the name of the form section from “Main” to “Personal details”. Then we double click on the section to edit it in the form section designer. We only leave the following attributes in this form section: “MemberNumber”, “FirstName”, “LastName”, “Address”, “EmailAddress”, “Gender”, “DateOfBirth”, “Status”. Then we click on the Save button and return to the “Form” editor. We now need to add another form section, so we click on the icon. We enter the name of the new form section “Payments” and double click on it to start form section designer. We leave the following attributes in the form section: “Number”, “FirstName”, “LastName”, “Loans” and “Reservations”. Note that we can use the same attributes in different form sections.

In a similar manner we create the form section called “Payments” with the following attributes “MemberNumber”, “FirstName”, “LastName”, “OutstandingCharges”, “Fees” and “Payments”. We will add the last section “Loan History” a little later.

We will now add the “Loan History” section. Note that we want to show only current loans in the “Current activity” section and we want to show all loans in the “Loan History” section. In fact, we want to show current loans by default in all forms of the Member object except the “Loan History” section of the “Editing” form. To specify that current loans should be shown by default in all forms of the Member object we select the “Loans” attribute of the Member object on the Attributes tab and click on the “Presentation” tab on the list of properties of the attribute Then we click on the “Record sorting/filtering” property to display the dialog. On this dialog we tick the “Filter” check box, select the “Custom query” radio button and click on the “Settings” button. We specify the single condition for the query: Loan.Status = ‘Current’.

Now we will open the “Editing” form of the Member object and add the new form section “Loan History”. We select the single attribute “Loans” to be present on this form section. We also want to override the default presentation of this attribute that shows only current loans to show all loans of a member. In order to do this we double click on the “Loan History” form section to open up form section designer and we click on the icon to switch from the “preview” mode to the “form layout” mode. We then select the “Loans” attribute on the layout of the form section and select “Record sorting/filtering” from the list of properties of the attribute. We un-tick the “Filter items” checkbox on the dialog that comes up to make sure that all loans are shown.

We have finished defining the multi-tabbed form – now every time the Member object is viewed/edited ***Aware IM*** will create a form with multiple tabs that a user can switch between.

4.4.3.3 Improving form layout.

As we said earlier by default Aware IM generates forms in a top-down fashion – one attribute underneath the other one. If a form has many attributes this layout may be

inconvenient – we may want to save vertical space by placing some attributes on the same row. We will show how to do this for the “Editing” form of the Member object. We will place the “LastName” attribute on the same row as the “FirstName” attribute.

Open the Member object for editing, expand the “Editing” form on the “Form” tab and double click on the “Personal details” form section to bring up the form section designer. Click on the  icon to insert a new column. You will see the blank column appearing on the form section. Click on the cell representing the “LastName” attribute in the left column to select it. Now click on it again and without releasing the mouse button drag it to the right column opposite the “FirstName” attribute. You can see now that the “LastName” attribute is located in the same row as the “FirstName” attribute. Click on the “Save” button to save changes to the form section.

Now the “LastName” attribute for the “Member activity” tab on the editing form of the Member object will be shown in the same row as the “FirstName” attribute. We can make similar changes to the “Payments” and “Personal details” form sections of the Member object as well as to forms of various other objects.

4.4.3.4 Presentation of simple attributes.

When ***Aware IM*** generates the default form it uses the default representation of attributes. This default presentation depends on the attribute type and other properties. For example, for attributes with choices when other values are not allowed it generates a drop down control, for attributes of the “Yes/No” type it generates a checkbox control etc. Sometimes it is desirable to use other types of controls for certain attributes – for example, for attributes with choices we could use radio buttons instead of a drop down control. ***Aware IM*** allows a certain level of customization in how attributes are displayed (see also the “Adding/Editing Attributes” section in the Aware IM User Guide).

In this section we will change the default presentation options of the “Gender” and “Status” attributes of the Member object. Both attributes have two choices (“Female” and “Male” for the “Gender” attribute and “Active” and “Suspended” for the “Status” attribute) and other values are not allowed. Because we have quite a bit of space on the form we will represent these choices as radio buttons, rather than a drop down.

Open the Member object for editing and select the “Gender” attribute on the “Attributes” tab. Click on the “Presentation” tab on the list of attribute properties. Select “Radio buttons” in the “Widget” property drop down. Then click on the “Save” button to save changes to the object. Similar changes can be applied to the “Status” attribute.

4.4.3.5 Presentation of reference attributes.

We will have a separate look at the presentation options for reference attributes (see also the “Presentation Options for References” section in the Aware IM User Guide).

Most of the time instances of the objects referred to through reference attributes are displayed on a form as a table, where every instance is represented as a row in this table. Every row shows certain attributes of the referred object. For example, loans

on the member form are shown as a table. For each loan this table shows when the loan was made and its due date. ***Aware IM*** automatically generates such a table and shows the default number of instances of the referred object (you can navigate to other instances as well). ***Aware IM*** picks the attributes of the referred object that it shows in this table at random.

Most of the time we want to customize the default appearance of the reference table and specify the attributes that will be shown. We will start by customizing the appearance of the "Loans" attribute of the Member object. When loans of a member are displayed we want to show the following details for each loan: loan status, item title and due date. We have "Status" and "DueDate" attributes in the Loan object, but the title of the item that has been borrowed is only available through the referred item: `Loan.Item.Title`. At this stage ***Aware IM*** cannot show such attributes in a reference table – it can only show attributes that are defined in the referred object itself. Fortunately, there is another option – we can define the attribute of the "Shortcut" type in the Loan object (called, for example, "ItemTitle") that will point to the "Title" attribute of the referred Item object. To do this open the Loan object for editing and define the new attribute called "ItemTitle". Its type should be "Shortcut" and the shortcut path should be "Item.Title" (the attribute will point to the "Title" attribute through the reference attribute of the Loan object "Item").

Now we can change the presentation of the "Loans" attribute in the Member object to show "Status", "ItemTitle" and "DueDate" attributes. To do this open the Member object for editing and select the "Loans" attribute. Click on the "Presentation" tab on the list of properties for the attribute. Then click on the "Widget settings" property. The dialog will be displayed. Tick the above attributes from the list of "Attributes to Display". Then click OK on the dialog. We will also sort loans in the reference table by status and due date, so we click on the "Record sorting/filtering" property and set the corresponding sorting criteria in the "Record Sorting" table. We will now click on the OK button to save the sorting dialog and then click on the Save button to save the changes to the Member object.

Similar changes should be made not only to the "Loans" attribute, but to all other references of the Member object and all references of other objects as well. You can check presentation of all references in the sample configuration located in the "samples" directory of your Aware IM installation.

Displaying references as a drop down

One of the references will be displayed a little differently so we will mention it here. We are talking about the reference attribute "Type" in the Item object. This attribute refers to the ItemType object. Unlike many other reference attributes this attribute is editable. This means that the user will be able to change it by selecting a different instance of the ItemType object from the list of all existing item types. When a reference attribute is represented as a reference table, if you want to add an instance to a table or select a different instance you need to click on a button that brings you to a separate screen that lists all existing instances. In this particular case, though, there are usually a very few instances of the ItemType object in the system so we want to show them all on the form itself (for example, as a drop down list, so the user

will be able to select the item type from the drop down without having to go to a separate screen).

To specify that the “Type” attribute of the Item object should be represented as a drop down list we open the Item object for editing, select the “Type” attribute on the “Attributes” tab and click on the “Presentation” tab on the list of attribute properties. Then we select the “Combo-box” option in the drop-down representing the “Widget” property (rather than the usual “Grid” option.)

Highlighting overdue loans

Another improvement that we will make to the presentation of loans on the form of the Member object is highlighting overdue loans. In the table of loans for a member entries representing overdue loans will be displayed in red, whereas all other loans will be displayed in the default color. To achieve this we will select the “Loans” attribute of the Member object, click on the “Presentation” tab and then click on the “Item display rules” property. This will show the “Item Rules” dialog that allows us to specify the conditions under which a particular entry will be highlighted.

An entry representing a loan should be highlighted if the loan is current and its due date is in the past:

```
  Loan.Status = 'Current'
AND
  Loan.DueDate < CURRENT_DATE
```

We need to enter these conditions into the table displayed by the dialog. Each condition is specified in the “Condition” column. Enter the above condition directly into the Condition column.

Now we need to define the properties of the entry in the table of loans that will be displayed when the above conditions are met. Click on the “Set Style” button. In the dialog that appears select “Use this color” radio button for the foreground color and pick the red color. Also select the “Use this font style” radio button and select “Bold” as font style. Click on the OK button to save the property and then click on the Save button to save changes to the object.

4.4.4 Improving operations

We will now improve the operations that we have defined for our application (see 4.3).

4.4.4.1 Invoking operations from reference tables

Operations can be invoked not only from the system menu or from the form of a business object or from the screen displaying query results, but also from tables of referred instances displayed inside business object forms. For example, the form of the Member object displays a list of member loans, where each loan represents an entry in this table. We want to be able to have a button next to this entry that will invoke the “Renew Loan” operation (see 4.3.10) and another button that will invoke the “Return Item” operation (see 4.3.11).

To achieve this we select the “Loans” attribute of the Member object and then click on the “Presentation” tab on the list of properties. Then we click on the “Widget settings” property. On the dialog that appears we click on the “Add” button located next to the “Operations with items” table. We specify the name of the operation “Renew”, select the type of the operation “Start Process” and select the process to be started “RenewLoan”. We click on the OK button on this dialog and define another operation called “Return” that starts the “CloseLoan” process.

In a similar fashion we will also add the “Cancel” operation starting the “CancelReservation” process to the “Reservations” attribute of the Member object (see 4.3.13).

4.4.4.2 Applicability conditions for operations

If an operation is not applicable under certain circumstances we may not want to show the button invoking the operation at all. For example, if an item is already on loan we do not want to show the “Borrow” button. Of course, even if the button was there and the user clicked on the button, she would still get the error message and no harm will be done (our business rules should take care of invalid situations irrespective of whether the button is shown or not). It would be better, however, if we could hide the button as well.

To achieve this we need to enter the applicability condition for the operation, in other words the conditions under which the operation is applicable. In the case of the item that is to be borrowed the condition should check whether the item is available:

```
Item.Available = 'Yes'
```

We need to enter this condition to the operations invoked from both the form of the Item object and from the screen showing results of the “Item” query (see 4.3.5). We open the Item object for editing and select its “Editing” form on the “Forms” tab. Then we click on the “Panel Operations” property in the list of properties of the form and then select the “Borrow” operation and click on the “Edit” button. We enter the above condition into the “Applicable when” text box. Now we open the “Item” query, click on the “Display as” property, select the “Borrow” operation in the list of “Operations with items” and perform similar changes there.

Other conditions that we will enter are:

- For the “Reserve” operation invoked from the form of the Item object and from the “Item” query we will add the following condition: `Item.Available = 'No'`
- For the “Return” operation invoked from the form of the Loan object and from the reference table of the “Loans” attribute of the Member object we will add the following condition: `Loan.Status = 'Current'`
- For the “Renew” operation invoked from the form of the Loan object and from the reference table of the “Loans” attribute of the Member object we will add the following condition: `Loan.Status = 'Current'`
- For the “Cancel” operation invoked from the form of the Reservation object and from the reference table of the “Reservations” attribute of the Member object we

will add the following condition: `Reservation.Status='Waiting' OR Reservation.Status ='Offered'`

4.4.4.3 Additional operations

We will add a couple of operations to our application to make it more convenient to users.

“Member Edit” operation.

Firstly we will introduce the “Member Edit” operation. This operation will ask the user to provide the details of the Member object she wants to edit and then displays the form of the object for editing. The only difference with the “Change Member Details” operation (see 4.3.3) is that if the user provided the details that uniquely identify a particular member, the form of the member will be shown immediately (with the “Change Member Details” operation the user will see the screen with the query results even if the query only found one member). This can only be achieved by the `PICK FROM` action of the rule language so we need to define a process called “EditMember” with the following actions:

```
PICK FROM Member NO CONFIRMATION FOR ONE
VIEW Member
```

The first action will run the “Member” query and will immediately proceed to the next action if one member was found without waiting for the user to confirm the member (if the query found more than 1 member the query result screen will be displayed as usual and the user will have to pick a member). The second action will show the editing form of the Member object.

We will define a new item in the system menu of the “Administrator” visual perspective (see 4.3.1) called “Edit” under the “Member” menu item.

“View Member Details” operation.

The next operation we will add will show all the main details of a particular member, such as her main personal details, a list of current loans, reservations and outstanding charges. Users will not be able to change these details. The goal of this operation is to provide a quick way for the user to view, but not change, all the important details of the member in one place. The operation will be available for members as well. It will be invoked from the system menu.

Before we can configure this operation we need to define a special form for the Member object that will only show the main details of a member. To do this we open the Member object for editing and click on the icon on the “Forms” tab. We will enter the name of the new form “Member details”. We will then save the changes to the object and double click on the form section underneath the new form to open the form section designer. We will add the following attributes to the “Main” form section in the form: “FirstName”, “LastName”, “MemberNumber”, “Address”, “EmailAddress”, “DateOfBirth”, “Loans”, “Reservations” and “OutstandingCharges”. We will arrange these attributes in two columns (see 4.4.3.3) and we will add separators between two groups of attributes – “Personal details” for the first 6 attributes and “Activity” for the

last 3. To add a separator click on the  icon in the form section designer toolbar. Then move the separator row to the desired place in the layout by pressing up and down arrows.

The operations of the new form should be the same as operations of the “Editing” form with the addition of the “Edit” operation that should bring up the editing form (add an operation of the “Edit” type and specify the name of the form – “Editing”).

Once we have defined the new form we can configure a new process that will bring it up. We should consider the fact that the operation will be used by members as well (see 4.4.2.1) so the new process “ViewMember” will have the following actions:

```
IF
  LoggedInSystemUser.AccessLevel='Member'
THEN
  VIEW LoggedInMember USING 'Member details' NOEDIT
ELSE
  PICK FROM Member NO CONFIRMATION FOR ONE
  VIEW Member USING 'Member details' NOEDIT
```

Note that “`NOEDIT`” keyword indicates that the values on the form can only be viewed, not changed. Note also that just like with the “Member Edit” operation we will display the form immediately if only one member has been found.

Now we only need to define the menu item invoking the operation in both the “Administrator” and “Member” visual perspectives. We define the “View details” menu item under the “Member” item in the “Administrator” perspective and the “Loans and reservations” menu item in the “Member” perspective.

4.5 Implementing other Requirements

As the final step in our configuration we will have a look at the requirements that we have left out initially.

4.5.1 Managing Communication with Members

The first of the remaining requirements is that our application has to manage communication with members. It has to automatically send offer e-mails or letters to members when the item they have reserved becomes available. Also the system has to be able to register all other forms of the communication with members – phone calls, discussions, faxes etc (see 3.2.7).

We start as usual by defining additional business objects and rules. The business objects that we identify are as follows:

- OutgoingLetter
- OutgoingEmail
- ContactNote

The meaning of the first two objects is obvious. The third object aims to capture other forms of communication with members, such as received letters, phone calls, faxes, discussion etc. E-mails received from members deserve a separate discussion. ***Aware IM*** is capable of receiving and registering e-mails automatically (and our application will include this feature – see 4.5.1.5). Whether this feature will be enabled, though, in the final system depends on the administrator of a particular library. If, for whatever reason, the feature will be disabled, then the system will still allow users to register incoming e-mails manually – through the use of the ContactNote object.

The attributes of the new business objects are as follows:

OutgoingEmail Object:

- *Subject* – text
- *Message* – text
- *CreatedOn* – timestamp (the date and time e-mail was created, calculated)
- *Status* (Sent, Unsent)
- *SentOn* – timestamp (the date and time the e-mail was sent, calculated)
- *SentToAddress* – text (the actual address the e-mail was sent to)
- *Attachment1*, *Attachment2*, *Attachment3* – documents (will allow up to 3 attachments)

References:

- *Correspondent* (single reference to a Member object to whom the e-mail was sent)

OutgoingLetter Object:

- *Subject* – text
- *Text* – text (text of the letter)
- *CreatedOn* – timestamp (the date and time letter was created, calculated)
- *Status* (Sent, Unsent)
- *SentOn* – timestamp (the date and time the letter was sent, calculated)
- *LetterDocument* – document (the actual letter)

References:

- *Correspondent* (single reference to a Member object to whom the letter was sent)

ContactNote Object:

- *Subject* – text
- *Type* – text (type of the contact – Phone Call, Discussion, Fax etc)
- *Notes* – text (description of the contact)
- *CreatedOn* – timestamp (the date and time the record about a contact was made, calculated)
- *ContactTime* – timestamp (the date and time contact with the member took place)
- *Attachment* – document (attachment if any)

References:

- *Correspondent* (single reference to a Member object with whom there was a contact)

4.5.1.1 Business Object Groups

In the previous section we did not focus on the relationship between Member and OutgoingLetter, Member and OutgoingEmail, Member and ContactNote. Should the Member object have a relationship with the OutgoingLetter, OutgoingEmail and ContactNote objects? Yes, it should because a member should know all the communication that took place with her. So in principle, the Member object could have a relationship with each of the above objects independently.

However, if we look carefully at OutgoingLetter, OutgoingEmail and ContactNote objects, we will notice that they all have a lot of things in common. From the Member object's perspective these are all different types of communication with the Member.

Maybe OutgoingLetter, OutgoingEmail and ContactNote should be represented by a single object (called, for example, Communication) where the type of communication is an attribute (just like it is in the ContactNote object)? This is certainly possible. However, we should notice that even though there are a lot of things that are common to all these objects, there are certain things that are specific to a particular object. For example, an e-mail may have "CC" and "BCC" attributes, not relevant to other objects, the state of the received e-mail can be "Read" and "Unread" whereas the state of the outgoing letter or e-mail can be "Sent" and "Unsent" and so on. In other words all these objects are quite different semantically.

Aware IM solution to this commonality problem is *business object groups* (see also the "Business Object Groups" section in the Aware IM User Guide). By making business objects members of a business object group we recognize the fact that business objects are sufficiently different (so the existence of them as separate objects is justified), yet there is a lot of commonality between them (this commonality is above all manifested by the existence of attributes of the same name and of the same type).

So we need to introduce a business object group called "Communication" and make OutgoingLetter, OutgoingEmail and ContactNote objects members of this group.

Once we introduce the Communication group, the Member object can be related to this group only rather than to each individual member of this group independently.

To summarize, we have introduced a number of new business objects with a relationship to the Member object and a new business object group Communication with a relationship between the Member object and the Communication group.

We can now configure the OutgoingLetter, OutgoingEmail and ContactNote objects (see 4.1.2). Note that at this stage we define only simple attributes of these objects without defining a reference to its Member object. The reason is that the matching attribute on the Member object must refer to a group and we do not have the group yet.

Defining Business Object Group

Now it is time to define the “Communication” business object group. We right click on the “Business Object Groups” node in the Elements Tree and select “New” in the pop-up menu – the editor of the business object groups appears in the working area. We specify the name of the new group “Communication”. Then we select the “OutgoingLetter” object from the panel containing available objects and click on the right arrow button to move it to a panel containing the group members. Then we do the same thing with the “OutgoingEmail” and “ContactNote” objects. Our group is ready now and we click on the “Save” button. The new group appears in the Elements Tree.

Defining Reference to a Group

The last step is to define a reference to the Member object in all members of the Communication group and define the matching reference to the Communication group in the Member object. We can do this by going to the Member object and adding the “Communication” attribute. Note that existing business object groups appear in the list of available attribute types just as individual business objects. So we select the Communication group as the attribute type, specify the name of the attribute “Communication”, tick “Multiple allowed” (member can have multiple communications), specify the relationship type as “Owner of” (communications cannot exist without the member they belong to) and create the matching attribute with the name “Correspondent”. Note that since the “Communication” attribute in the Member object refers to a group, the system automatically creates matching attribute in all members of the group! If we open each member of the Communication group now we will see the “Correspondent” attribute referring to the Member object. The only problem is that the system automatically created the attribute as “Multiple allowed” so we need to un-tick this flag[11].

4.5.1.2 Business rules

We now need to define business rules applicable to the new objects.

OutgoingEmail object.

There are a number of calculated attributes defined for this object, so we need to calculate values of these attributes. The “Sent on” rule calculates the value of the “SentOn” attribute:

```
IF
  OutgoingEmail.State WAS CHANGED TO 'Sent'
THEN
  OutgoingEmail.SentOn=CURRENT_TIMESTAMP
```

We also need to calculate the “SentToAddress” attribute, so we define the “Destination address” rule:

```
IF
  OutgoingEmail.State WAS CHANGED TO 'Sent'
```

[11] Note that we could create the relationship between the Communication group and the Member object from any of the members of the Communication group by defining the Correspondent attribute referring to the Member there and creating the matching attribute on the Member object.


```
THEN
  OutgoingEmail.SentToAddress =
OutgoingEmail.Correspondent.EmailAddress
```

Once e-mail has been sent we shouldn't allow anyone to change the "SentToAddress", so we need to protect this attribute:

```
IF
  OutgoingEmail.State='Sent'
THEN
  PROTECT OutgoingEmail.SentToAddress FROM ALL
```

OutgoingLetter object

We only have one rule that calculates the "SentOn" attribute:

```
IF
  OutgoingLetter.State WAS CHANGED TO 'Sent'
THEN
  OutgoingLetter.SentOn=CURRENT_TIMESTAMP
```

4.5.1.3 Sending outgoing e-mails

According to the requirements the application is supposed to automatically send e-mails or letters to members when the items they have reserved become available and to remind them that their loan is overdue (see 3.2.3). In order for the system to be able to send e-mails to members, the Member object must become *intelligent* (see also the "Intelligent Business Objects" and "Outgoing Email" sections in the Aware IM User Guide).

Making Member object intelligent.

To make the Member object intelligent open the Member object for editing and click on the "Communication" property in the list of object properties. Tick the "Email" channel" in the list of channels for the object. This means that it is possible to communicate with this object through e-mails. Select the "Use values from SystemSettings object" radio button (we will explain this object in 4.5.3.1). Click on the OK button on the dialog and click on the "Save" button to save the changes to the object.

Defining e-mail notifications.

The next step is to configure a notification representing an e-mail. Note that unlike most business objects notifications are not stored in the system – they just represent a piece of information that is sent around or exchanged. So there is a difference between the OutgoingEmail object, which registers all e-mails sent to members and an e-mail notification, which represents the information sent to members. If we did not want to register e-mails we would not need the OutgoingEmail object, yet we would still need the email notification.

We will define the following notifications:

1. A notification representing an e-mail that is sent when a reservation is offered

2. A notification representing an e-mail that is sent to remind a member about overdue loan
3. A notification representing any other e-mail that may be sent to a member by an operator at any time

Let us define the first notification. To do this, right click on the “Notifications” node in the Elements Tree and select “Add Outgoing Email Notification” from the pop-up menu. ***Aware IM*** will automatically create a new notification with the attributes required for the notification to be an e-mail. The editor for this notification will appear in the working area of the screen. We will change the name of the notification from “OutgoingEmail” to “ReservationOfferEmail” and then select the “Subject” attribute. In the list of attribute properties we will enter the following text as the initial value for this attribute “Reserved item available”. Now we will add the initial value for the “Message” attribute. As the text of the initial value is quite large we will click on the “Initial value” property and enter the text in the dialog that appears:

```
Dear <<Reservation.Member.FirstName>>
<<Reservation.Member.LastName>>,

The item <<Reservation.Item.Title>> you reserved on
<<Reservation.ReservedOn, dd/MM/yyyy>> is now available for
you to borrow. You have 7 days to borrow the item. The
reservation expires on <<Reservation.ReservedOn+7*24,
dd/MM/yyyy>>.

Regards,
The Library.
```

Note that we assume that the Reservation object that we refer to in the tags of this message is available in the context.

In a similar fashion we define the “LoanOverdueEmail” notification – see the sample application located in the “samples” directory of your Aware IM installation for the text of the message. We also define the “OutboundEmail” notification without any initial values for the “Subject” and “Message” attributes – this notification will represent a general e-mail sent by an operator.

Add business rules.

Let us now add business rules that will send the notifications we have defined. We will start with the reservation offer e-mail. Offered reservations are handled by the rules of the Reservation object (see 4.2.4). We will add the “Reservation offer e-mail” rule that will send the “ReservationOfferEmail” to a member when the state of the reservation changes to “Offered”:

```
IF
  Reservation.Status WAS CHANGED TO 'Offered'
AND
  Reservation.Member.EmailAddress IS DEFINED
THEN
```



```
  SEND ReservationOfferEmail TO Reservation.Member
  CREATE OutgoingEmail WITH
    OutgoingEmail.Message=ReservationOfferEmail.Message,
    OutgoingEmail.Correspondent=Reservation.Member,
    OutgoingEmail.Subject=ReservationOfferEmail.Subject,
  OutgoingEmail.State='Sent'
```

The `CREATE` action in the rule above creates the OutgoingEmail object so that the e-mail is registered in the system. Note that the Reservation object that the ReservationOfferEmail expects in the context *is* in the context, since this is the object the rules of which are being executed.

Now we need to add a business rule to the Loan object that will send the "LoanOverdueEmail" notification to the member (see 4.2.3):

```
IF
  Loan.OverdueFeeChargedOn WAS CHANGED
AND
  Loan.Member.EmailAddress IS DEFINED
THEN
  SEND LoanOverdueEmail TO Loan.Member
  CREATE OutgoingEmail WITH
    OutgoingEmail.Text=LoanOverdueEmail.Message,
    OutgoingEmail.Correspondent=Loan.Member,
    OutgoingEmail.Subject=LoanOverdueEmail.Subject,
  OutgoingEmail.State='Sent'
```

4.5.1.4 Sending letters

If a member does not have e-mail (which means that there is no value for the "EmailAddress" attribute) the application should automatically prepare a letter that informs the member that the reserved item became available or reminds the member about overdue loan. Also when the library sends any other letter to a member (for whatever reason) we want our system to automatically prepare a standard template of a letter that has library details and logo, so that the operator only fills in the text of the letter.

All outgoing letters must be registered with the system, so the system must create the instance of the OutgoingLetter object. The actual document containing the electronic copy of the letter will be stored in the "LetterDocument" attribute of the OutgoingLetter object.

Defining document templates.

We start by defining the templates of the letter documents we will be sending. We will create the following document templates:

1. Reservation offer letter
2. Loan overdue letter
3. Letter to member template

The last one is the general template of a letter to a member. All templates will be created as MS Word documents; however, if a target system does not have MS Word we will also prepare the alternative version of the documents in the plain text format. ***Aware IM*** will automatically use the alternative versions if MS-Word is not available or not supported on the target platform, provided that we follow the naming conventions where the name of the alternative document is the same as the name of the original document followed by the "_Alternative" postfix (see also the "Adding/Editing Document Templates" in the Aware IM User Guide).

We can prepare the document templates in MS Word and then import them into ***Aware IM***. To import the document create the document of the MS Word type (see 4.4.1), click on the "Import" button on the documents editor pane and select the document file. You can have a look at the prepared templates in the sample configuration of the library application located in the "samples" directory of your Aware IM installation. Note that the "Reservation offer letter" template assumes that the Reservation object is in the context and uses it inside its tags; the "Loan overdue letter" assumes that the Loan object is in the context and the "Letter to member" template assumes that the OutgoingLetter object is in the context.

Business rules.

Now we need to add business rules to prepare reservation offer letter and reminder letter if a member does not have e-mail address. We add the "Reservation offer letter" business rule to the Reservation object (see 4.2.4):

```
IF
  Reservation.Status WAS CHANGED TO 'Offered'
AND
  Reservation.Member.EmailAddress IS UNDEFINED
THEN
  CREATE OutgoingLetter WITH
  OutgoingLetter.Subject = 'Reservation offer letter',
  OutgoingLetter.Correspondent = Reservation.Member,
  OutgoingLetter.Text = 'See document',
  OutgoingLetter.LetterDocument = 'Reservation offer letter'
```

The initialization of the "LetterDocument" attribute in the above rule is especially interesting. This attribute of the "Document" type is initialized with the name of the document template. ***Aware IM*** will automatically replace the contents of the tags used in the template with values of attributes of the OutgoingLetter object and store the resulting document in the "LetterDocument" attribute (see also the "Document Generation" section in the Aware IM User Guide).

We also need to add the "Reminder letter" business rule to the Loan object (see 4.2.3):

```
IF
  Loan.OverdueFeeChargedOn WAS CHANGED
AND
```



```
  Loan.Member.EmailAddress IS UNDEFINED
THEN
  CREATE OutgoingLetter WITH
  OutgoingLetter.Subject = 'Item overdue',
  OutgoingLetter.Correspondent = Loan.Member,
  OutgoingLetter.Text = 'See document',
  OutgoingLetter.LetterDocument = 'Loan overdue letter'
```

Finally we need to make sure that when the operator prepares a letter to a member the letter is created according to the format of the “Letter to member template”. When the operator prepares the letter the system will be creating an instance of the OutgoingLetter object. The operator will be entering the text of the letter in the “Text” attribute of the OutgoingLetter object. The entered text will be transferred to the template (if you look at the template you will see the <<OutgoingLetter.Text>> as the text of the letter). We only need to make sure that the “LetterDocument” attribute of the OutgoingLetter object is set to the name of the template. In fact, we can do this by setting the initial value of this attribute to the name of the “Letter to member” template. So all we have to do is set the initial value of the “LetterDocument” attribute to “Letter to member template”.

4.5.1.5 Handling incoming e-mails

We can get our application to automatically receive and register incoming e-mails that arrive to a certain e-mail address (our library will advise members to send their e-mails to this address).

To do this, double click on the node representing the business space version in the Elements Tree and click on the “Incoming Emails” property in the property list of the version. The “Incoming e-mails” dialog will be displayed. Select the “The system will handle incoming e-mails” radio button, select the “Use values from SystemSettings object” radio button (we will look at this object in 4.5.3.1) and click on the OK button. The “Automatic Elements” dialog will be displayed. Tick the “I want to automatically register all incoming e-mails” checkbox and click on the OK button.

If we now look at the Elements Tree we will see that the Configuration Tool automatically added the “IncomingEmail” notification representing the incoming e-mail and the “ReceivedEmail” business object to store received e-mails. It has also added the rules that create and initialize the instance of the “ReceivedEmail” object when an incoming e-mail is received (see also the “Incoming Email” section in the Aware IM User Guide).

As the “ReceivedEmail” object represents communication with a member we will add this object to the Communication business object group and add the following attributes to this object:

- Correspondent (reference to the Member object that we received the e-mail from)
- State (with choices “Read” and “Unread” with the initial value “Unread”)

We will also modify business rules automatically created by ***Aware IM*** to register incoming e-mails to set the value of the "Correspondent" attribute. To do this we open the "IncomingEmail" notification for editing and go to the "Rules(received") tab at the bottom. We see the automatically created rule in the rule editor. We add the action that finds the member by the "from" address and we modify the "`CREATE`" action to initialize the "Correspondent" attribute with the found member. The resulting rule looks like this:

```
FIND Member WHERE
  IncomingEmail.FromAddress CONTAINS Member.EmailAddress
CREATE ReceivedEmail WITH
  ReceivedEmail.Correspondent=Member,
  ReceivedEmail.Subject=IncomingEmail.Subject,
  ReceivedEmail.Message=IncomingEmail.Message,
  ReceivedEmail.SentOn=IncomingEmail.SentDate,
  ReceivedEmail.ToAddress=IncomingEmail.ToAddress,
  ReceivedEmail.FromAddress=IncomingEmail.FromAddress,
  ReceivedEmail.CC=IncomingEmail.CC,
  ReceivedEmail.BCC=IncomingEmail.BCC,
  ReceivedEmail.Attachment1=IncomingEmail.Attachment1,
  ReceivedEmail.Attachment2=IncomingEmail.Attachment2,
  ReceivedEmail.Attachment3=IncomingEmail.Attachment3
```

4.5.1.6 Operations

We will add the following operations to our system menu (in the "Administrator" visual perspective) that will deal with member communication:

- "Email to member" item in the "New" folder – allows creating and sending arbitrary emails to members
- "Letter to member" item in the "New" folder – allows creating and sending arbitrary letters to members
- "Member contact note" item in the "New" folder – creates ContactNote object
- "Correspondence" item in the "Find" folder – searches the system for all e-mails, letters and contact notes with the specified details
- "Review unsent letters" item in the "TODO List" folder – searches the system for all letters with "Unsent" status
- "Print unsent letters" item in the "TODO List" folder – prints all letters with "Unsent" status
- "Review unread e-mails" item in the "TODO List" folder – searches the system for all received e-mails with "Unsent" status

Configuration of most operations is straightforward and is similar to the configuration of other operations that we have already covered in 4.3. You can check this in the configuration of the sample library application located in the "samples" directory of your Aware IM installation.

The only item that we want to mention here is "Correspondence" that should run a query to find communication with a member. This query searches the business object

group, not the individual business object as before (see 4.3.2, for example). Conditions of such queries can only involve attributes common to all members of the group (they must have the same name and type) and the query can only display common attributes. As we wanted to check the text of the communication we made sure that all members of the group have “Message” attribute (we ended up renaming the “Text” attribute in the OutgoingLetter object to “Message”, for example). We also wanted to display the names of the member that the communication was with. At the moment it is not possible to include reference attributes into the list of attributes displayed by a query (although it is possible to include them in query conditions), so we added shortcuts to all members of the Communication group (“CorrespondentFirstName” and “CorrespondentLastName”) – see 4.4.3.5

4.5.1.7 User interface

To wrap up the implementation of “communication management” feature we need to make improvements to the user interface to have different forms for entry and editing where necessary, allow operations on reference attributes etc (see 4.4.3). We can do all these improvements in a similar fashion to what we did with other objects.

You can check the sample configuration of the library located in the “samples” directory of your Aware IM installation

4.5.2 Payment by credit card

Another feature that we have initially left out is online payment by members (see 3.2.6). We will have a look at how to configure this feature in this section.

In our application members will pay through the PayPal credit card payment system. In order to pay through PayPal one has to have a PayPal account. We will assume that our library has opened such an account. Technically, any system that wants to accept payments by credit card made through PayPal has to call the URL of the PayPal site and pass a number of parameters, such as the name of the account, product quantity, price, product code and URL of the page to return to. The rest is then handled by secure PayPal pages where the customer enters her credit card number and finalizes the purchase. If the purchase is successful PayPal returns to the URL you specified as “success” URL. If the purchase failed PayPal also returns to the URL you specified as “failure” URL.

Aware IM makes it possible to easily integrate credit card payment into your application. We will show you how to do this in the next sections.

4.5.2.1 New business objects

First of all we need to configure a business object that will represent the PayPal credit card payment system. To do this we define a new business object as usual (see 4.1.2), enter its name “PayPal” and enter one attribute “Name”[12]. Our application will be communicating with the PayPal system, therefore we click on the

[12] In principle, PayPal object does not need any attributes. However, Aware IM requires that at least one attribute is defined for a business object.

"Communication" property and ***Aware IM*** displays a number of possible channels that it can communicate with.

The channel that we will be communicating through is called the *URL channel* – see also the "Defining Communication Channels" section in the Aware IM User Guide. This channel allows communicating with a software system through URL. We tick the checkbox in the row representing the URL channel we specify the settings of the channel. We enter "https://www.paypal.com/cgi-bin/webscr" as the URL of the service provider – this is the URL of the PayPal page handling the payment. Then we enter "return" as the name of the parameter indicating the URL to return to if a service has been successful. This is the name of the parameter that PayPal expects (among other parameters passed to the PayPal page), which contains the URL that PayPal will return to if payment has been successful. We do not have to worry about the URL itself – this is taken care of automatically by ***Aware IM***. Similarly we enter the name of the parameter indicating the URL to return to if service fails – "cancel_return". We click on the OK button and then we click on the "Save" button to create the PayPal object.

Now we need to define the object that will represent parameters that PayPal expects when its URL is called. PayPal expects the following parameters:

- "business" – contains the name of the PayPal account
- "amount" – purchase quantity
- "item_name" – the name of the item being purchases
- "item_code" – internal code of the item being purchased
- "currency_code" – code of the currency
- "no_shipping" – if value is "1" shipping information is not included
- "cmd" – code of the PayPal operation. Must be "_xclick" in our case.

We define the new business object called "PayPalParameters" with the attributes that have exactly the same names as the parameters above. All attributes can be of the "Plain Text" type. We specify the following initial values for the attributes:

Attribute	Initial value
Business	The name of the PayPal account of our library
Amount	1
item_code	0010 (can be anything we like)
currency_code	USD
no_shipping	1
Cmd	_xclick

Once we have defined the PayPalParameters object we need to define the service of the PayPal object. Intelligent objects can expose services that we can request from rules. We will define such a service for the PayPal object. We open the PayPal object for editing and click on the "Communication" property in the list of properties of the object. Then we click on the "Add" button to add a new service. The "Service" dialog is displayed. We enter the name of the service "PayByCreditCard" and select the

PayPalParameters object as input for this service. Then we click on the OK button and then save changes to the PayPal object.

4.5.2.2 Changes to the Payment object

When a member will be making payment by credit card our application will be creating an instance of the Payment object, just like it does when the operator registers payment of a member (see 4.3.15). Unlike the registration of payment operation, though, there will be a potentially long period when a member makes a payment through PayPal pages. During this period the Payment object will be in an intermediary state, since we don't know yet whether the payment will succeed or fail. Therefore we introduce a life cycle for the Payment object – we add the "Status" attribute to this object with possible values "New" and "Applied". The "Applied" state is assigned to the object after payment has been completed[13].

We modify the "Payment amount validation" rule to consider the "New" state:

```
IF
  Payment.Status='New'
AND
  Payment.Amount > Payment.Member.OutstandingCharges
Then
  REPORT ERROR 'Payment amount cannot exceed outstanding
charges.'
```

Once payment has been made we do not want anyone to change its details, therefore we enter the "Payment protection" rule:

```
If
  Payment.Status='Applied'
Then
  PROTECT Payment FROM ALL EXCEPT System
```

We also add the action that sets the Payment state to "Applied" in the "RegisterPayment" process (see 4.3.15)

4.5.2.3 Credit card payment operation

In our credit card payment operation we will be creating the instance of Payment object and then use the details of this object to call service of the PayPal object. So we need a process to perform these actions. We define the new "MakePayment" process that does not require any input[14] and has the following actions:

```
ENTER NEW Payment WITH Payment.Member=LoggedInMember
CREATE PayPalParameters WITH
  PayPalParameters.amount=Payment.Amount,
  PayPalParameters.item_name=Payment.Description
```

[13] We can also have the "Failed" state which indicates that the payment has failed

[14] We do not need to provide the Member object as process input since the process will only be used by members and therefore we will use the currently logged in member


```
REQUEST SERVICE PayByCreditCard OF PayPal USING
  PayPalParameters
Payment.Status = 'Applied'
DISPLAY MESSAGE 'Your payment has been successfully processed'
```

The actions above are quite straightforward. The only point we want to mention here is that we do not create the instance of the PayPal object in the process nor do we need to worry about the PayPal object being in the context. We will create only one instance of the PayPal object and we will do it when the system is initialized (see 4.5.3), not inside the process. In fact, if we are dealing with an intelligent business object that supports the URL channel only one instance of this object should exist in the system. Consequently, it is not necessary to find this instance to put it in the context – ***Aware IM*** will do it automatically.

Now we can define the “Pay by credit card” menu item in the “Member” visual perspective. This item will start the “MakePayment” process. Our members can now pay by credit card online.

4.5.3 System initialization

The final topic that we want to cover regarding configuration of our application is initialization of the application. When configuring an application we need to consider whether we are configuring it for a particular library or it will be used by many different libraries. In the latter case every library that will be using the application may need to perform initialization of the application specific to that library. For example, every library may use its own logo, name and address in documents that the library will be sending its members; the library may allocate a specific e-mail address to receive e-mails from members; the library may need to specify the name of the server that will be handling outgoing and incoming e-mails etc.

We want system administrator of the library to specify all these values once before the application is used by operators and members (she may also want to occasionally change these values during the operation of the system). We will show in this section how to support this initialization in the configuration.

4.5.3.1 System Settings object

In ***Aware IM*** there is one predefined object called “SystemSettings” that is responsible for storing the application initialization values (see also the “Setting Initial Values of the Information System” section in the Aware IM User Guide). ***Aware IM*** creates this object automatically for any business space. The system administrator may create an instance of this object once and provide values for the attributes of this object.

The SystemSettings object by default contains attributes that allow storing values for automatic handling of outgoing and incoming e-mails. When we have been defining properties of the Email channel and incoming e-mail handling (see 4.5.1.3 and 4.5.1.5) we have specified that the properties should be taken from the attributes of the SystemSettings object. This allows us to let the system administrator enter these values in the Operation Mode rather than “hardwire” them in the configuration.

We can also add our own attributes to the SystemSettings object. For our library application we will add the following attributes:

- *LibraryName* (Plain Text) – the name will be used in document templates and e-mails
- *LibraryAddress* (Plain Text) – the address will be used in document templates and e-mails
- *LibraryTelephoneNo* (Plain Text) – the number will be used in document templates and e-mails
- *MaxLoans* (Number) – this number identifies maximum number of loans that a member may have at any given time. From time to time the administrator may change this number
- *PayPalAccountName* – the name of the PayPal account of the library (see 4.5.2)

We will add these attributes to the SystemSettings object. Now we need to make changes to the document templates, e-mail notifications, rules and processes so that we use the attributes of the SystemSettings object, rather than fixed values. Note that it is not necessary to find the instance of the SystemSettings object to put it in the context, since there is always only one instance of the SystemSettings object in the system.

In particular, we will change the “Total item validation” rule of the Loan object (see 4.2.3) to use the “MaxLoans” attribute:

```
If
  Loan.Status='New'
AND
  COUNT Loan WHERE (
    Loan IN ThisLoan.Member.Loans
    AND
    Loan.Status='Current')
  >= SystemSettings.MaxLoans
THEN
  REPORT ERROR 'Member already has maximum allowed number of
items.'
```

We will also change the “MakePayment” process (see 4.5.2) to use the name of the PayPal account (we will only change the `CREATE` action):

```
CREATE PayPalParameters WITH
  PayPalParameters.business=SystemSettings.PayPalAccountName,
  PayPalParameters.amount=Payment.Amount,
  PayPalParameters.item_name=Payment.Description
```

4.5.3.2 Initialization process

Now we need to make sure that we let the system administrator initialize the SystemSettings object when the system is started for the first time. Note that if the

values of the SystemSettings object are not provided operators and members will not be able to use the system.

We will define the process that will create the instance of the SystemSettings object and let the user specify the values for this instance and we will configure the application to use this process on startup.

The new process will have the following actions:

```
DISPLAY MESSAGE 'Library database has not been initialized.
The system will now ask you to provide the initial data.'
ENTER NEW SystemSettings
CREATE PayPal
DISPLAY MESSAGE ‘Library system has been successfully
initialized’
```

The first action will notify the user about the initialization process, the second one will let the user enter the values of the SystemSettings object, the third action will create the instance of the PayPal object so that members can make credit card payments (see 4.5.2) and the last action informs the user about the successful initialization.

The only problem is that we do not want this process to run if the application has already been initialized. Therefore, we define the wrapper process that checks the existence of the SystemSettings object:

```
IF NOT (EXISTS SystemSettings) THEN Initialize2
```

Our first process will be called “Initialize2” and the second process will be called “Initialize”.

Now we only need to specify that the application should use the “Initialize” process at start-up. To do this we open the “Administrator” visual perspective for editing. Select the “Initialize” process in the dropdown of the “Initialization process” property in the list of properties of the visual perspective. Click on the “Save” button to save changes to the visual perspective.

We have now finished the configuration of our library application. We can now test the application as described in 4.3.17 and deploy it in production.

4.6 Operating the final system

If you want to start the library application server on the same machine that you are performing the configuration on you need to “publish” the business space version that you have configured. To do this right click on the node representing the business space version in the Elements Tree and select “Publish” in the pop-up menu. Once the business space version has been published its state will change to “`CURRENT`” which indicates that the application can be used in operation.

To start operating the system users will need to point their browsers to the following URL:

http://serverName:8080/AwareIM/logon.html[15]

where “serverName” is the name of the machine where Aware IM Server (or Aware IM Control Panel) is running. This name can be “localhost” if you are running it on one machine only or it can be the network name of your server if you are running the system only within your local area network or it can be the DNS name or IP address of the server in the Internet if you are using the system over the Internet.

If you have configured your application for other companies to use, then you need to prepare a distributable version of ***Aware IM*** that includes your configuration. In this case you do not need to “publish” the business space version. Instead you need to right click on the node representing the version you want to distribute and select “Build executable” from the pop-up menu - note that you will need a special ***Aware IM*** edition for developers to be able to use this feature. Once you select this feature ***Aware IM*** will display a dialog where you will be able to specify the name of the product, copyright and other information. When you click on the OK button ***Aware IM*** will build the executable in the directory you specify. You can distribute this executable to your users.

[15] Note that this URL should be used only if “Library” is the default business space, that is, the business space you specified when you installed ***Aware IM***. If “Library” is not the default business space you can use the following URL: http://serverName:8080/AwareIM/logonAdmin.html and specify the name of the business space explicitly.

