MASTER IN COMPUTER SCIENCE

From Organization Requirements to System Requirements a Library System Case Study

Nigot, Sylvie

Award date:
1995

Awarding institution:
University of Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique

From Organization Requirements to System Requirements: a Library System Case Study

Sylvie Nigot

Promoteur: Eric Dubois

Mémoire réalisé en vue de l'obtention du grade de Licencié et Maître en informatique

Année académique 1994-1995

Je souhaite remercier ceux qui ont contribué de près ou de loin à la réalisation de ce mémoire et je tiens plus particulièrement à exprimer ma gratitude envers

Monsieur E. Dubois pour son aide précieuse

M. Petit et l'équipe ICARUS pour sa disponibilité

E.Yu et J. Mylopoulos pour leur acceuil chaleureux ainsi que leur encadrement au cours de mon stage

From Organization Requirements to System Requirements: a Library Case Study

Abstract

The requirements engineering activity consists in two levels: one of them aims at specifying "what" has to be done while the other trys to describe "why" things are done the way they are.
Through the application of the Albert language and the i* (prononced istar) framework to the main functionalities of a library system, we show that the two levels interaction can be extremely efficient in order to achieve a better understanding along with an improvement of the process. Unfortunately, if the cooperation of the two frameworks seems efficient, an automatic binding between them still looks improbable.

Résumé

L'activité d'ingénierie des besoins consiste en deux niveaux: l'un s'attachant à la spécification du "quoi", l'autre s'attardant d'avantage à la modélisation du "pourquoi" des choses.
Au travers de l'application du langage Albert et de l'approche i* (prononcée istar) aux fonctionnalités d'une bibliothèque, nous montrons que l'interaction de ces deux niveaux peut se révéler extrêmement efficace en vue d'une meilleure compréhension et d'une amélioration sensible du système. Si la cooperation entre ces deux approches semble donner de bons résultats, la possibilité de l'établissement d'un lien systematique entre elles reste encore improbable.

Table of contents

Chapter 1: Introduction

Requirements Engineering and Process modelling activities constitue crutial steps in an Information System development process. Indeed, these activities allow an analyst to reach a full understanding of a work process and this, at different levels.
On one hand, the requirements specifications will provide a description of **what** a system is supposed to do and on the other hand, Process modelling will lead to the comprehesion of **why** this system is needed.
In this thesis, we will studied two frameworks (the Albert language and the i* framework), each of them "belonging" to one of these to levels.

In chapter 2, we will describe the Albert language and illustrate its main features through the example of a grocer's shop system. We will see its mechanisms of agents decomposition and declaration along with the constraints specifications.

In chapter 3, we will present the i* framework by first describing the Strategic Dependency (SD) model followed by the Strategic Rationale (SR) model. Once again, we will illustrate these models features through an example.

Chapter 4 will represent the heart of this thesis. After having specified in an informal way the main functionalities (book acquisition policy and loan policy) of a library case study, we will apply both approches (Albert and i*) in order to redesign the library system. The specification procedure will be the following:

- formal specification of the existing system using the Albert language
- organization modelling of the library using the SD model
- Research of alternatives and improvement of the work process by the mean of the SR model
- modelling of the new organization with an SD model
- formal specification of the resulting library system

In the last section of this chapter, we will initiate a little analysis in order to respond the question of knowing if an Albert specification could automatically induce the corresponding i* model and vice versa.

Chapter 2: The Albert language

2.1. Introduction

« ...Requirements Engineering is the statement of desired functional and performance characteristics of a software system independently of any actual realization... » [Dub86]

The Requirements Engineering activity is a critical step in the development of information systems and softwares. It consists in a work process specification in order to reach a precise and complete problem statement. it is thus crucial to cope with functional and non-functional requirements.
« NFRS define global constraints on a software system, such as development costs, operational costs, performance, reliability,...Should not be confused with functional requirements, which impose requirements on the function of a system.[Mylopoulos95]

In order to represent requirements, different trends can be identified: in one hand, we have different languages based on mathematical and logical theories. [Bub80] [Mylopoulos90]
On the other hand, based on an object-oriented paradigm, we propose to present in this chapter, the Albert[1] language developed within an ESPRIT II project called ICARUS. ([DDDP94a], [DDDP94b], [DDP93], [YDDM95])

This framework supports the requirements engineering of composite systems within organizations. By composite systems, we mean systems composed of heterogeneous components. The specification won't be limited to the software developed and will rather take into account the environment in which this system will be embedded. It includes, hardware pieces, humans, etc.

The Albert language consists of concepts and models that focus on understanding the "whats" underlying the requirements engineering activity. It will include the description of the set of functionalities necessary to achieve the organization's goal.

[1] Agent-oriented Language for Building and Eliciting Requirements for Real-Time systems

2.2. The language constructs

2.2.1. Introduction

Basically, the formal language is based on a mathematical language, the temporal logic, suited for describing histories. This logic is itself an extension of mutli-stored first order logic, still based on the concept of variables, predicates and functions. Three extensions are taken into account :

- the introduction of actions;
- the introduction of agents together with their properties (responsibilities for actions, for providing perceptions,...). This object-oriented concept can also be seen as a possible way of constructing large specifications in terms of more finer pieces, each of them corresponding to the specification of an agent guaranteeing a part of a global behaviour of the whole system;
- the identification of typical patterns of constraints which support the analyst in writing complex and consistent formulas. In particular, typical patterns of formulas are associated with actions.

Using the language involves two activities:

- writing declarations in order to introduce the vocabulary of the considered application;
- expressing constraints, i.e. logical statements which allow the dinstinction between possible behaviours of the system and unwanted ones.

A graphical syntax (with a textual conterpart) is used to introduce declarations and to express some typical constraints frequently encountered. The expression of the other constraints is purely textual.

2.2.2. Exemple: the grocer's shop system

Throughout this chapter, we will apply the Albert features to a simple example about a grocer's shop and its clients.

Description of the case

Clients go to a grocer's shop. They choose different items in the shop, and put them in their trolley. When they have finished, they go and present their items to the cashier. The latter will calculate the bill and present it to the client who will pay it. The grocer has to remove the money from the till each time the amount is superior to a given limit and has to put it in security. He/she is also responsible for the contents of the grocer's shop and items in the "fresh products" category can't stay in the shelves for more than two weeks.

2.2.3. Declarations

In the specification of composite systems, the declarations consist in the identification of the agents together with their states structure and the list of the actions. Importation and exportation links between agents are also graphically described.

Declaration of a society or agents hierarchy

A composite system specification can rapidely become very big, that's why, in order to reduce this complexity, it becomes very useful to group agents into societies. These societies can themselves be grouped together to form larger societies. Actually, the agents are organized in terms of a hierarchy where we distinguish between:

- Complex agents (made of finer agents);
- Terminal agents which can no further be decomposed

.

A society have no own structure nor behaviour. Only the behaviour of an individual agent will be formaly specified.
The whole system is of course considered as an agent society.

Graphically

A society is represented by an ellipse containging smaller ellipses. Multiple agent classes, made of several instances, are represented by cascaded ellipses. Each ellipse is labelled by the agent class identifier.

The Grocer's shop example

The graphical declaration associated with Grocery's System described earlier, is depicted figure 2.1.

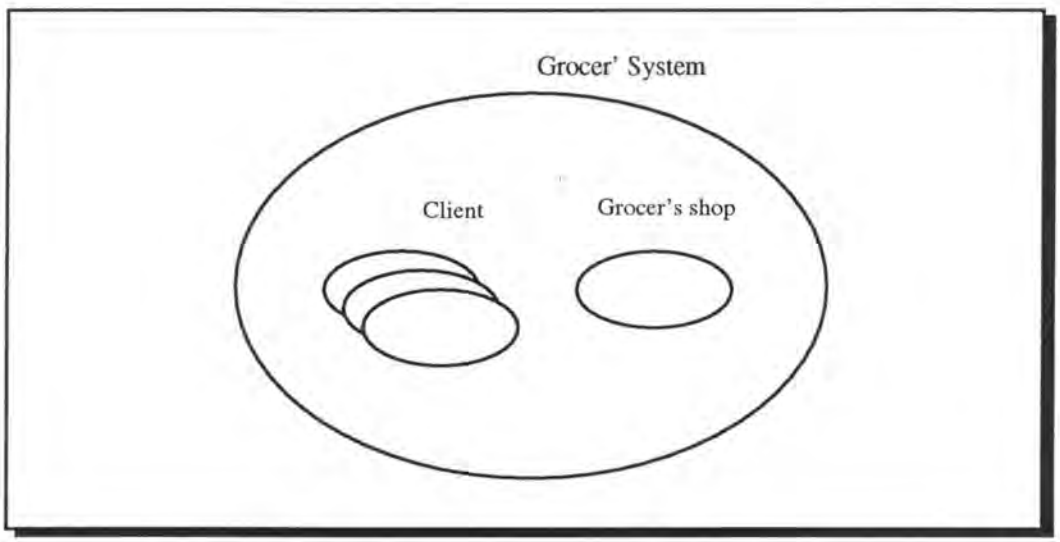


Figure 2.1 : Graphical declaration of the Grocer's System

The society *Grocer's* system is composed of two terminal agents: A Client (represented by a multiple agent class) and the Grocer's shop.

Declaration of an agent

The declaration part of an agent consists in the description of its state structure and the list of actions happening along its history.

The structure of a state is defined in terms of entities which can be grouped in populations or be individuals, values which are used for characterising attributes of entities and relationships between entities. On top of these usual concepts, the Albert language also uses data types which correspond to:

- predifined data types (STRING, BOOLEAN, INTEGER,...) equipped with their usual operations;
- more complex types built by the specifier using a set of predifined type constructors like Set, List, Cartesian Product,...
- elementary types which are defined by the user. For that specific kind of type, there is no associated structure;
- specific types corresponding to agent identifiers. For example, each *Client* agent has an identifier with a CLIENT type.

Actions have arguments belonging to data types.

Graphically

A state component is represented by a box labelled by the state component identifier with a rectangle inside (or two linked rectangles for a table). This rectangle indicates the type of the elements of the state component.
We have different boxes according to the type of the represented state component. See figure 2.2.
As shown in figure 2.2(e), a derived component is linked by a broken arrow to the state component from which it is derived.

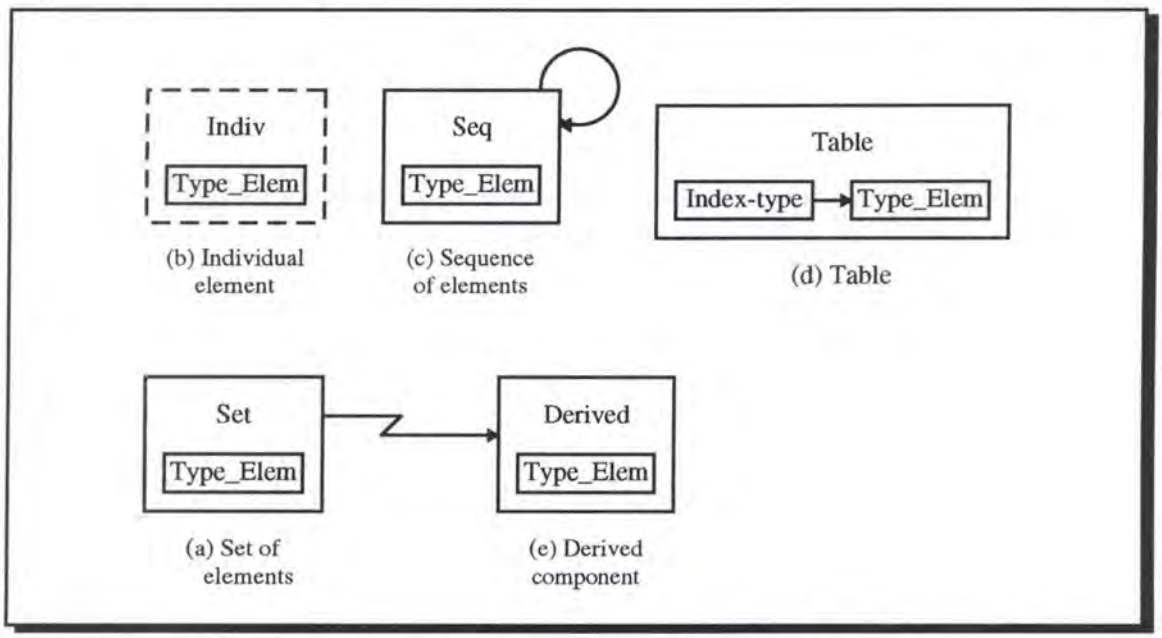


Figure 2.2: graphical representations of state components

An action is represented by an oval inside a box. The box is labelled by the name of the action. Arguments can characterize actions. As we saw, all arguments are typed and their types are linked to the box with an arrow starting from the action box.
Figure 2.3, the action *Action_3* has one argument of type *Type_Arg*.

Importation and exportation of elements

Diagrams also include graphical notations making possible to distinguish between internal and external actions and state components, and to express the visibility guaranteed by the agent to the outside.
The internal structure of an agent is represented by a parallelogram and information within this parallelogram is under the control of the described agent.

- Inside the parallelogram, boxes without dotted arrow (for a state component or an action), indicate that this information remains private and therefore, won't be seen from the outside (for a state component), or won't have any effect on other agent's behaviour (for an action).

  In figure 2.3, which summarizes globaly the graphical representation of an agent declaration, one can see that the action *Action_1* will only affect the behaviour of the concerned agent. Equally, the state component *Table* will only be perceived by this agent.

Conversely, boxes with arrow(s) denote state components which are exported to the indicated agent(s). The action *Action_2* will be exported to *Agent_2* and *Agent_3* while the *Set* element will be only perceived by *Agent_2*.

- Information outside from the parallelogram denotes elements which are imported from other agents. *Action_3* is from the *Agent_3* initiative but will affect the *Agent_1*'s behaviour.

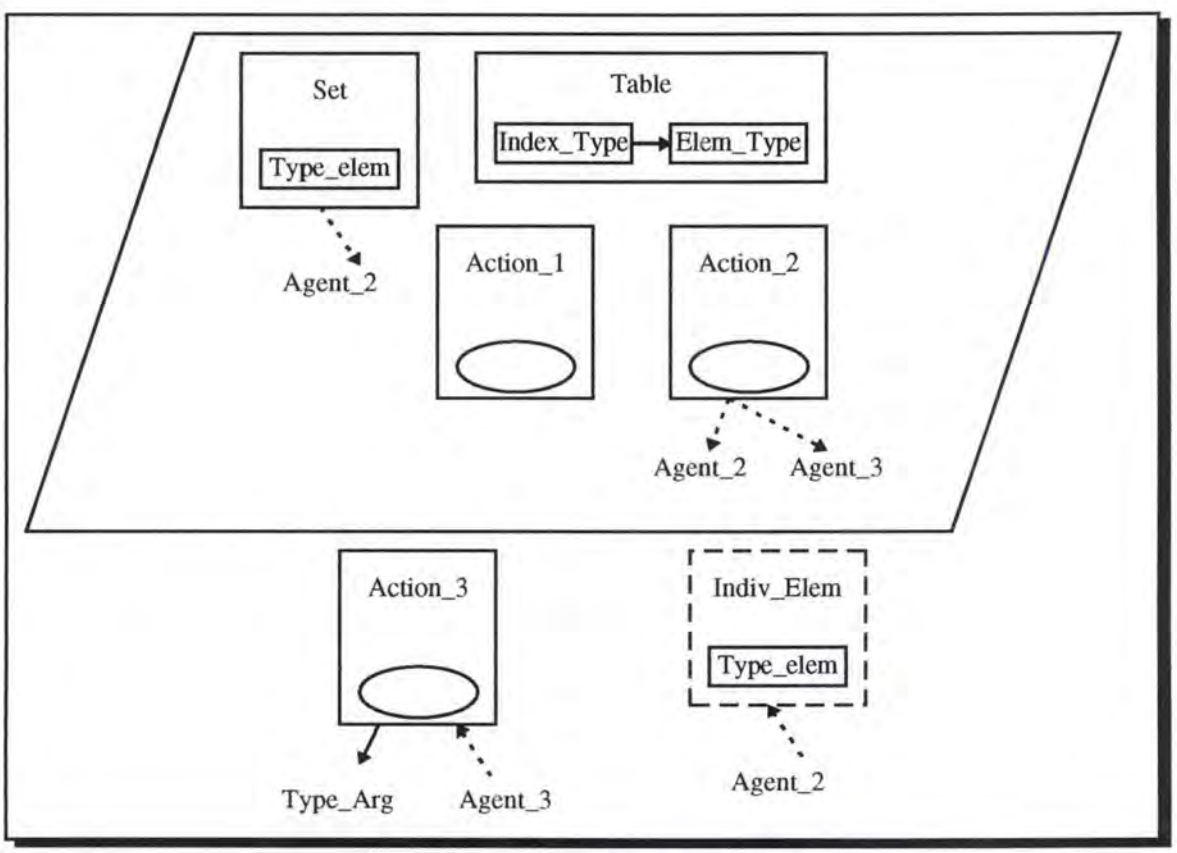


Figure 2.3 : Graphical declaration of agent_1

The Grocer's example

Figure 2.1. shows us two terminal agents: the Client and the Grocer's agents.
Their respective graphical representations are shown in figure 2.4, figure 2.5.

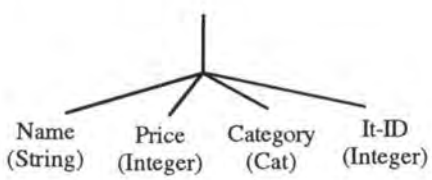
Declaration of the Client agent

From the graphical declaration depicted on figure 2.4, it can be read that the Client state is made of:

1. Two internal state components (Trolley and Status_C)

Trolley: This component represents the set of the items (item) the client choosed in the grocer's shop.

- The type *Item* can be described by:



Each item found in a client's trolley is characterized by its name, its price, a category and an identification.

- The type *Cat* is defined by:

Cat = {Fresh products, tinned food, drinks}

Status_C: Once a client has entered the grocer's shop, he/she has a status (Status).

- The type *Status* is defined by:

Status = {shopping, Presenting, has paid}

A client is shopping or he/she's presenting his/her trolley to the grocer or he/she has already paid for the goods.

2. One external state component (Items_Shop).

Items_Shop: it represents the set of items (Item) available in the shop.

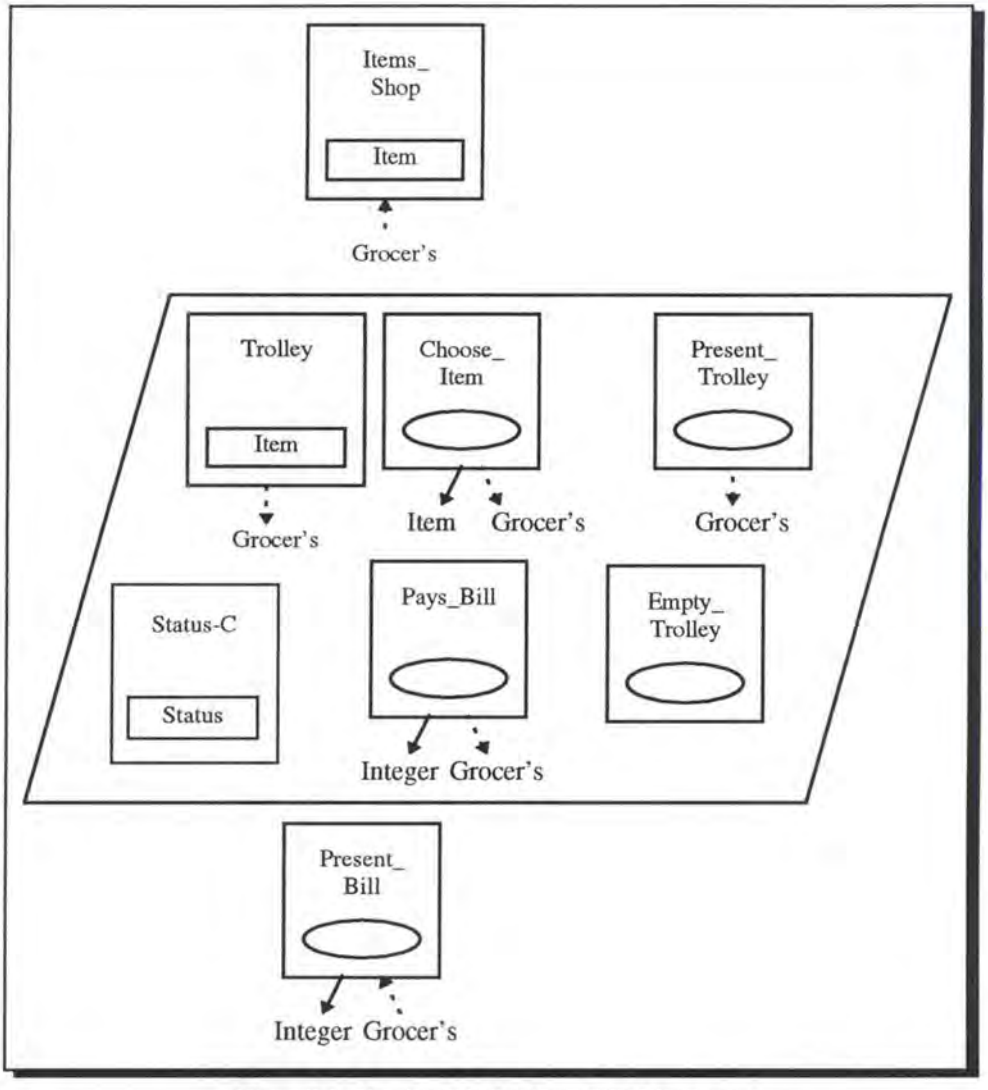


Figure 2.4 : Graphical declaration of the Client agent

3. Four internal actions (Choose_Item, Present_Trolley, Pays_Bill and Empty_Trolley) for which the Client has the initiative.

Choose_Item: the client agent choose an item (Item) proposed in the grocer's shelves, and put it in its trolley.

Present_Trolley: the client presents his/her trolley to the grocer's in order to pay the different items.

Pays_Bill: the agent pays the bill (Integer) to the grocer's.

Empty_Trolley: the client empties his/her trolley

4. One action (Present_Bill) perceived by the Client has an external initiative.
Present_Bill: The Grocer's gives to the client the bill (Integer) for all its buyed items.

Declaration of the grocer's agent

The graphical declaration depicted on figure 2.5 is related to the Grocer's shop structure. We'll only describe the new elements. From this figure, it can be read that:

1. The Grocer's agent has one external state component (Trolley).

2. Three actions are issued by the Grocer's agent (Present_Bill, Remove_M_Till and Remove_Item).
Remove_M_Till: the grocer removes the money present in the till and puts it in security.
Remove_Item: the grocer removes an item from the grocer's shelves.

3. this agent perceives three external actions (Present_Trolley, Choose_Item and Pays_Bill) from the Client initiative.

4. Finally, this agent has four internal state components (Items_Shop, Till, Limit and Remove_Time).
Till: this individual element represents an amount of money present in the till at a certain time.
Limit: it represents the maximum amount of money allowed to stay in the till
Remove_Time: this state component is derived from the limit and the till components and it indicates when it's time to remove the money from the till.

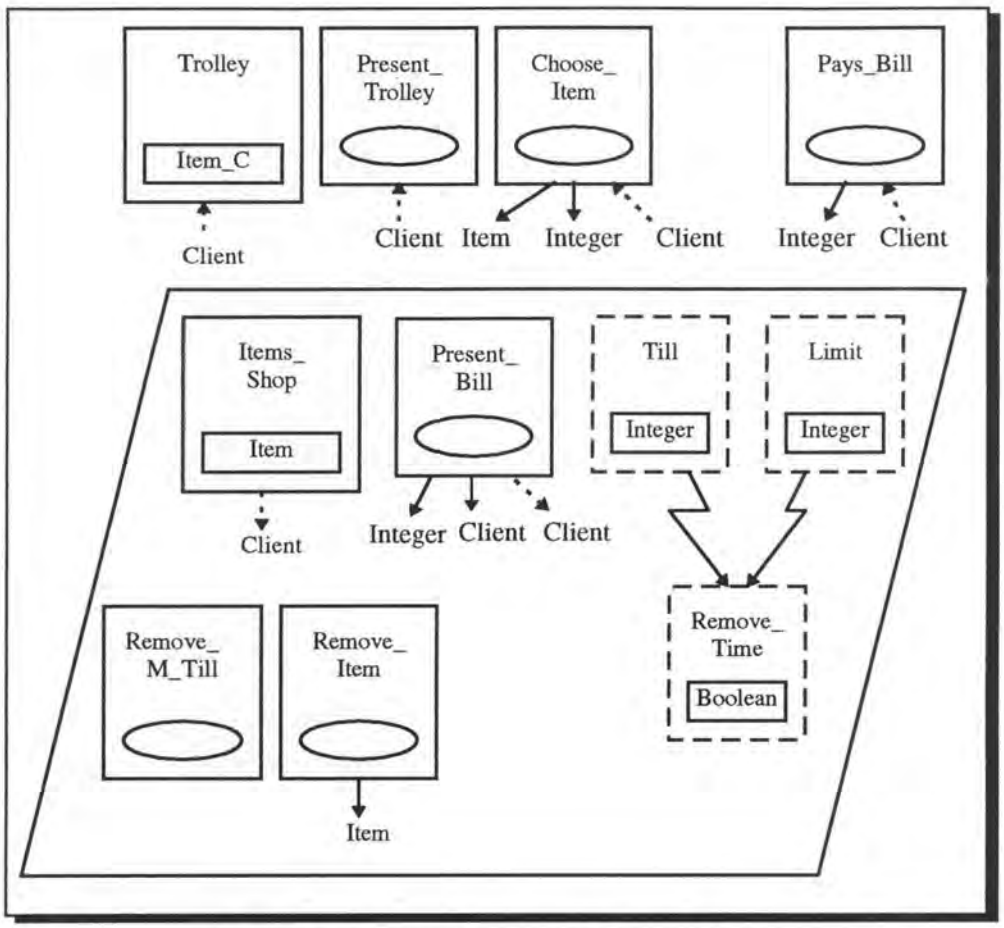


Figure 2.5 : Graphical declaration of the Grocer's agent

2.2.3. Constraints

Each agent is defined by a set of possible lives[2] limited by the expression of different kinds of constraints.

Their expression is formal and based on multi-sorted first order logic. They use the concept of variables, predicates and functions. The different constraints are grouped into three families: Basic Constraints, Local constraints and Cooperation Constraints.

The complete formal specifications associated with the Grocer's shop and the Client agents have been described at the end of this chapter.

Basic constraints

Basic constraints are used to describe the initial state of an agent and to give the derivation rules for the derived components.

[2] A life is an (in)finite sequence of sates and actions. Each state (structured in terms of entities) is labelled by a time value which increase all along the life. Actions occur between two successive states and can be simultaneous. Constraints are used for purning the (usually) infinite set of lives [DDDP94a].

- The **derivation rules** will give the relationship between derived components values and the state components values from which they are derived.

Remove_Time $\triangleq$ Till $\geq$ Limit
** It's time to remove the money from the till when the amount equals or is upper than the maximum limit.*

- The **intitial valuation** groups the constraints describing the initial states of the agent life.

Trolley = { }
This means that the client's trolley is empty at the beginning of the system life.

Local constraints

Local constraints are related to the internal behaviour of the agent. They are classified under four headings: state behaviour, effects of actions, causality and capability.

- **State behaviour**

Constraints under this heading express properties of the states or properties linking states in an admissible life of an agent.
First of all, there are constraints which are true in all states of the possible traces of an agent. These constraints are written according to the usual rules of strongly types first order logic.
They are formed by means of logical connectives: $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\Rightarrow$ (implies), $\Leftrightarrow$ (if and only if), $\forall$ (for all), $\exists$ (there exists).

On top of constraints which are true in all states (usually referred as invariants), there are constraints on the evolution of the system (like, e.g. if this property holds in this state, then it holds in all future ones) or referring states at the different times. Writing these constraints requires to be able to refer to more than one state at a time. This is done in the language by using additional temporal connectives which are prefixing statements to be interpreted in different states. These connectives are inspired from temporal logic:
if Ψ and φ are statements:

- $\Diamond \varphi$: φ is true sometimes in the future (including the present);
- $\blacklozenge \varphi$: φ is true sometimes in the past (including the present);
- $\Box \varphi$: φ is always true in the future (including the present);
- $\blacksquare \varphi$: φ is always true in the past (including the present);
- $\varphi \mathcal{U} \Psi$: φ is true from the present until Ψ is true (strict);

- $\varphi \mathcal{S} \Psi$: φ is true back from the present since Ψ was true (strict).

There are constraints related to the expression of real-time properties. There are needed to describe delays or time-outs (like, e.g., « an element has to be removed from its population within 15 minutes ») and are expressed by subsripting temporal connectives with a time period. This time period is made precise by using usual time units: sec, min, hours, days,...

Cat (it) = {fresh product} $\Rightarrow \neg \Box_{>15\ days}$ In(Items_Shop, it)
** An item of the 'fresh product' category can't stay for more than 15 days in the grocer's shelves.*

- **Effects of actions**

Under this heading, we describe the effects of actions which may alter states in lives. Only actions which bring a treacable change are described here.
In the decription of the effect of an action, we use an implicit frame rule saying that states components for which no effect of actions are specified do not change their value in the state following the happening of a change.
The effect of an action is expressed in terms of a property characterising the state which follows the occurrence of the action. The value of a state component in the resulting state is characterised in terms of a relationship reffering to:
1. the action arguments;
2. the agent responsible for this action (if this action is an external one, the name of the agent is prefixing the action);
3. the previous state of the history.
In the pattern associated with the definition of an action, the left hand side of the equation characterises the state as it results from the occurrence of the action while the right hand side refers to the state as it is before the occurrence of the action.

Choose_Item(it): Trolley = Add(Trolley, it)
** Each time a client chooses an item in the grocer's shop, it puts it in its trolley, increasing the number of the choosen items.*

- **Causality**

This heading is related to the causality relationship existing between some occurences of actions.
Expressing causality rules with usual temporal connectives may appear very cumbersome. To this end, the language is enriched with specific connectives which allow to specify, for example, that an action has to be issued by the agent as a unique response to the occurrence of another action (brought or not by the agent). A common pattern is based on the use of the « $\rightarrow$ » symbol which is not to be comfused with the usual « $\Rightarrow$ » logical symbol.
The « $\rightarrow$ » symbol can be quantified by a temporal operator to express performances constraints.
The right part of a commitment (the reaction) may only refer actions which are issued by the agent (i.e. actions which are not prefixed).

Left and right parts of a commitment may be composed of one or more occurences of actions. In case of more than one, occurences may be composed in the following ways:
1. « act1;act2 » which means « an occurrence act1 followed by an occurrence act2 »;
2. « act1 $\otimes$ act2 » which means « an occurrence act1 and an occurrence act2 (at the same time) »;
3. « act1 $\|$ act2 » which means « an occurrence act1 and an occurrence act2 (in any order);
4. « act1 $\oplus$ act2 » which means « an occurrence act1 or an occurrence act2 (exculsive or).

Some more complex expressions are provided to express iterative applications of actions.

Client.Present_Trolley $\rightarrow$ Present_Bill(i, Client)
** When a client presents his/her trolley, he/she receives the bill for the chossen items*

- **Capability**

Under this heading, is described the role of the agent with respect to the occurrence of its own actions. To this end, we are still using an additional extension of the classical first-order and temporal logic by making possible to express permissions associated with an agent. We then consider three specific connectives allowing the expression of obligations, preventions and exclusive obligations (respectively the $\mathcal{O}$, the F and the $\mathcal{XO}$ connectives).
The pattern for an obligation « $\mathcal{O}$ (<int-action>/<situation>) » expresses that the action has to occur if the circumstances expressed in the situation are matched (these circumstances refer to conditions on the current state).

The pattern for a prevention « F (<int-action> / <situation>) » expresses that the action is forbidden when the circumstances expressed in the situation are matched.
The pattern « $\mathcal{XO}$ (<int-action>/<situation>) » is used to express exclusive obligation, it is a shorthand for the combination of « $\mathcal{O}$ (<int-action>/<situation>) » and « F (<int-action> / ¬ <situation>) ».
The default rule is that all actions are permitted whatever the situation.
Using the connectives makes possible to express the control that the agent has with respect to its internal actions.

F(Remove_Item/ Status ≠ "has paid")
** A client can only remove the items from his/her trolley if he/she has already paid for them.*

Cooperation constraints

This family of constraints specifies how the agent interacts with its invironment, i.e. how it perceives actions performed by other agents (action perception), how it can see parts of the state of other agents (state perception), how it lets other agents know what actions it does (action information) and how it shows parts of its state to other agents (state information).
Perception and information provide the specifier a way to add a dynamic dimension to the importation and exportation relationship between agents expressed in the declaration part of the specification.

- **Action perception**

Beyond this heading, Albert defines how the agent is sensitive to changes occuring in its environment, which are made available to it by other agents belonging to the same society.
Action perception are specified using the K (knowledge), I (ignorance) and $\mathcal{X}$K (exclusive knowledge) connectives.

The pattern «K (<ext-action> / <situation>) » defines the situation where, if an action is issued by external agent, the behaviour of the current agent is influenced.

The pattern «I (<ext-action> / <situation>) » defines the situation where, if such action is issued by external agent, it has no influence on current agent's behaviour.

The pattern « $\mathcal{X}$K (<ext-action> / <situation>) » is used to express exclusive obligation, it is a shorthand for the combination of «K (<ext-action> / <situation>) » and «I (<ext-action> / ¬ <situation>) ».

The default rule is that all imported actions available may be perceived whatever the situation.

$\mathcal{X}$K(_.Present_trolley/TRUE)
** The grocer's always knows when a client presents hit/her trolley.*

- **State perception**

Beyond this heading one defines how the agent sees part of the state of other agents belonging to the same society and which are made available to it by them. State perceptions are also specified using the K, I and $\mathcal{X}$K connectives.
The default rule is that all imported state components available may be perceived whatever the situation.

$\mathcal{X}$K(grocer's.Items_Shop/TRUE)
** the client always knows the contents of the grocer's shelves.*

- **Action information**

Constraints under this heading specify how occurences of actions performed by an agent are made available to other agents belonging to the same society. This is also a dynamic property and is expressed using the K, I and $\mathcal{X}$K connectives introduced above.

The pattern « K (<int-action>.<agent> / <situation>» defines the situation where occurences of an internal action are made available to a given agent.

The pattern « I (<int-action>.<agent> / <situation> » defines the situation where the occurences of an internal action are not made visible for a given agent.

The pattern « $\mathcal{X}$K (<int-action>.<agent> / <situation>» is used to expressed exclusive obligation, it is a shorthand for the combination of « K (<int-action>.<agent> / <situation>» and « I (<int-action>.<agent> /¬ <situation> ».

The default rule is that all exported actions may be visible by any agent to which it is exported, whatever the situation.

XK(Present_Bill(i, c1).c2 / c1 = c2)
** the grocer can only present a bill to the corresponding client*

- **State Information**

Beyond this heading is defined how the agent shows parts of its state to other agents belonging to the same society. State information is also specified using the K, I and $\hat{K}$ connectives.
The default rule is that all exported state components may be visible by any agent to which it is exported, whatever the situation.

$\hat{K}$(Trolley.grocer's/Status = "presenting")
** Means that the grocer's only sees the content of the trolley when the client has presented his/her trolley*

Constraints of the Client agent

BASIC CONSTRAINTS

- **Initial valuation**

Trolley = { }
** The client's trolley is empty at the beginning of the system life.*

Status = {shopping}
** When a client enters the grocer's shop, he/she's there to do some shopping.*

LOCAL CONSTRAINTS

- **Effects of actions**

Choose_Item(it): Trolley = Add(Trolley, it)
** Each time a client chooses an item in the grocer's shop, it puts it in its trolley, increasing the number of the choosen items.*

Present_Trolley: Status = "presenting"
** When a client presents his/her trolley to the grocer, he/she's not shopping anymore.*

Pays_Bill(i): Status = {Has paid}

Empty_Trolley: Trolley = { }
$\wedge$ Status = "shopping"

- **Causality**

Grocer's.Present_Bill(i) → Pays_Bill(i) ; Empty_Trolley
** Each Present_Bill order issued by the grocer's, is followed by a Pays_Bill action occurrence followed by an Empty_Trolley action.*

- **Capability**

F(Remove_Item/ Status ≠ "has paid")
F(Choose_Item/ Status ≠ " shopping")

COOPERATION CONSTRAINTS

- **State Perception**

$\mathcal{K}$K(grocer's.Items_Shop/TRUE)
** the client always knows the contents of the grocer's shelves.*

- **State Information**

$\mathcal{K}$K(Trolley.grocer's/Status = "presenting")
** Means that the grocer's only sees the content of the trolley when the client has presented his/her trolley*

Constraints of the Grocer's agent

BASIC CONSTRAINTS

- **Derivation rules**

Remove_Time $\triangleq$ Till ≥ Limit
** It's time to remove the money from the till when the amount equals or is upper than the maximum limit.*

LOCAL CONSTRAINTS

- **State behaviour**

Cat (it) = {fresh product} $\Rightarrow \neg \Box_{>15\ days}$ In(Items_Shop, it)
** An item of the 'fresh product' category can't stay for more than 15 days in the grocer's shelves.*

- **Effects of actions**

Remove_M_Till: Till = 0
** After a Remove_M_Till action occurrence, the till is empty.*

Remove_Item (it) : Items_Shop = remove(Items_Shop, it)
** The item it isn't available anymore after having been removed by the grocer.*

Client.Choose_Item(it): Items_Shop = remove(Items_Shop, it)
** The item it isn't available anymore after having been removed by the client.*

Client.Pays_Bill(i): Till = Till + i
** When a client pays his/her bill, the money is added to the content of the till.*

- **Causality**

Client.Present_Trolley $\rightarrow$ Present_Bill(i, Client)
** When a client presents his/her trolley, he/she receives the bill for the chossen items*

- **Capability**

F(Present_Bill(total, client)/ total $\neq$ IterAssoc(client.Trolley, price(i) + price(j))
** The grocer can't present a bill to a client if the bill isnt correct. The bill must correspond to the sum of the prices of the items contained in that client's trolley.*

O(Remove_M_Till/Remove_Time)
** The grocer must react when the amount limit of money present in the till has been reached. However, the grocer can decide to remove the money even if the limit is not yet reached.*

COOPERATION CONSTRAINTS

- **Action Information**

XK(Present_Bill(i, c1).c2 / c1 = c2)
** the grocer can only present a bill to the corresponding client*

- **Action Perception**

𝒳K(_.Present_trolley/TRUE)
** The grocer's always knows when a client presents hit/her trolley.*

I(_.Choose_Item(it)/Quantity(Items_Shop(i) with item(Items_Shop(i)) = it) = 0)
** The gorcer's ignores the action of choosing an item in the shelves when this particular item isn't available anymore.*

- **State Perception**

𝒳K(_ .Trolley/TRUE)
** The grocer's always sees the content of the clients*

- **State Information**

𝒳K(Items_Shop. _/TRUE)
** The shelves content is always shown to the clients.*

Chapitre 3: the Istar framework

3.1. Introduction

In order to improve the quality of software products, it is necessary to enhance the processes used to develop and maintain these products. It is also important to have a good understanding of the work process context in which such softwares are being developed.
Customers demands and the current competitive context induce many companies to reorganize their business processes and to adapt or rethink completely their actual technologies and organization according to their new objectives. This tendance is underlined by the reengineering activity. "At the heart of reengineering is the notion of discontinous thinking, of recognizing and breaking away from the outdated rules and fundamental assumptions that underlie operations."[Hammer90]
In its paper "Reengineering work: don't automate, obliterate", Hammer proposes several principles helping to find out brand new solutions to work processes problems instead of simply automatize the whole business:

- Organize around outcomes, not tasks
- Cature information once and at the source
- Link parallel activities instead of integrating their result

Such advises can only be taken into account if the analyst succeded to capture organizational intentions. "How can we re-engineer this process if we don't have information on the intentions of players in the process..." [Mylopoulos95]

Most of the models existing in the litterature aimed at describing "what" a work process is doing but these models merely express "why" things are like they are. They offer little help to find out these new alternatives boosted by the business reengineering activity.

In this chapter, we will present a new approach , the istar framework developed at the university of Toronto. The purpose of this framework is to support process modelling in the reenigineering activity. It has already been presented in different contexts such as:
information systems requirements engineering [Yu93b]
business process reengineering [Yu93a], [Yu94a], [Yu94c]
software processes [Yu94b]

The latest version of the complete framework has been developed in [Yu94]

The istar framework is composed of two models, the Strategic Dependency and the Strategic Rationale models. The first one, described in section 3.2, models the network of dependencies existing between the actors of a business process while the Strategic Rationale model, described in section 3.3, aimes at supporting the reasoning hidden behind a particular way of working.
These two models have a formal counterpart represented in the conceptual modelling language Telos [Mylopoulos91]. This part won't be developed in this chapter, for more details, see [Yu94].

3.2.1. Modelling features

We will now describe the main features of the Strategic Dependency Model and illustrate the different notions through different examples and mostly about an assurance system called "mutuel".
The SD Model is represented by a set of nodes and links. Each node can be assimilated to an actor of the process, and the links are the dependencies existing between them.
The SD Model proposes four types of dependency links: The goal, the task, the resource and the softgoal dependencies. The actor who depends on another actor is called the depender, the actor who is depended upon is called the dependee and the element for which the depender depends on the dependee is called the dependum. Graphically, the dependency between two actors is depicted on figure 3.1.

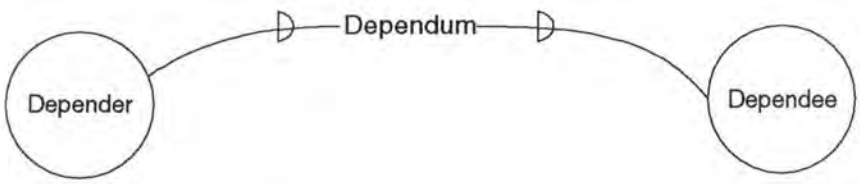


Figure 3.1: Dependency link between two actors

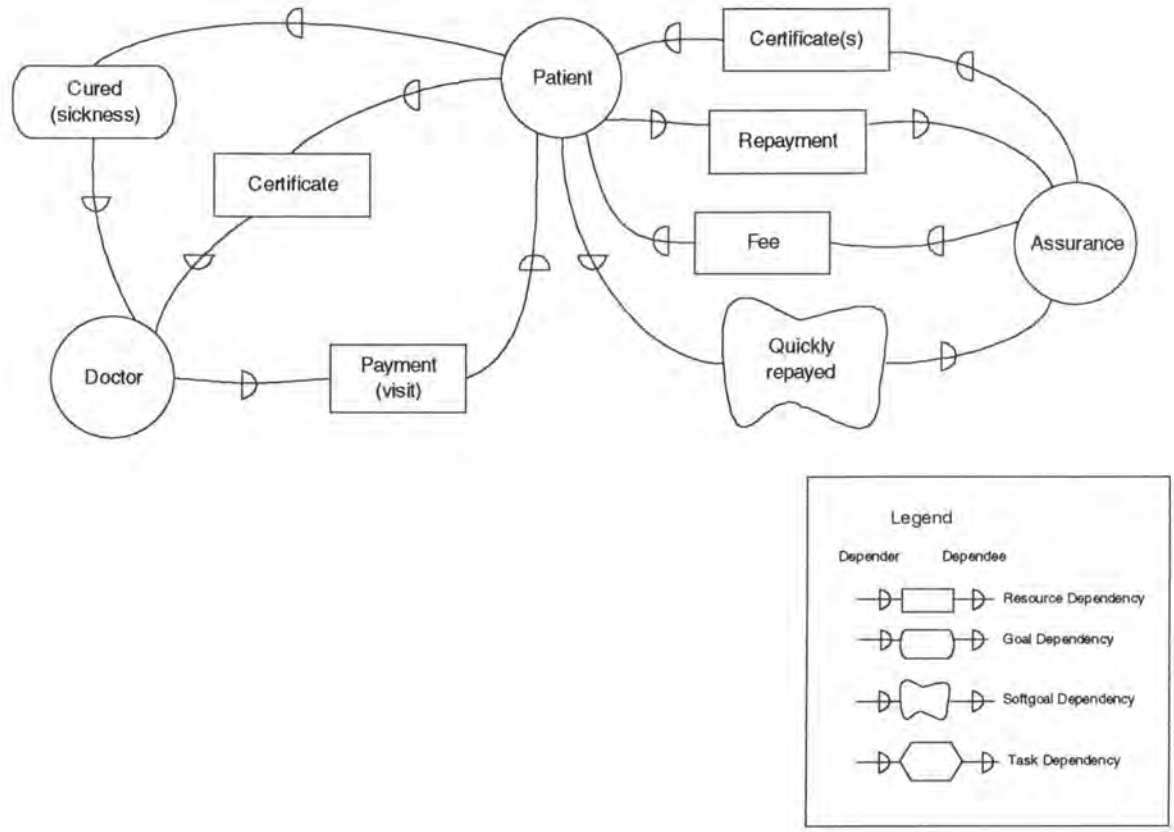


Figure 3.2: SD Model in a medical assurance domain

Figure 3.2. shows the Strategic Dependency Model of a special kind of medical assurance called « Mutuel ». It presents some of the relationships among patients, doctors and the assurance system.

A patient depends on a doctor for the treatment of his/her sickness. The doctor depends on the patient for the payment of the visit. In exchange, the doctor gives the patient a certificate for a partial repayment from the assurance.

The patient has also to pay a cotisation to the assurance company.

Dependency types

We said the SD Model made the distinction between four types of dependencies, based on the type of the dependum. We get three types of intentional dependencies: resource dependency, task dependency, and goal dependency. The fourth type, called softgoal dependency, is based on a notion of non-functional requirements (or quality requirement) in software engineering.

- "In a **goal dependency**, the depender depends on the dependee to bring about a certain state in the world. The dependee is given the freedom to choose how to do it. With a goal dependency, the depender gains the ability to assume that the condition or state of the world

will hold, but becomes vulnerable since the dependee may fail to bring about that condition."

In figure 3.2., the goal dependency "cured(sickness)" tells us that the patient depends on the doctor for the treatment of the sickness, but he/she doesn't know how the doctor will achieve this goal. Only the result counts for the patient and the doctor has all the necessary freedom to succeed.
Let's take another exemle developed in [Jacobson], about a telephone communication between two subscribers. A subscriber (called A_Subscriber) depends on the Exchange to get a connection with another subscriber (B_Subscriber). We have a goal dependency because the A_Suscriber doesn't know how the exchange will achieve this; all she/he wants is to be able to communicate with the B_Subscriber.

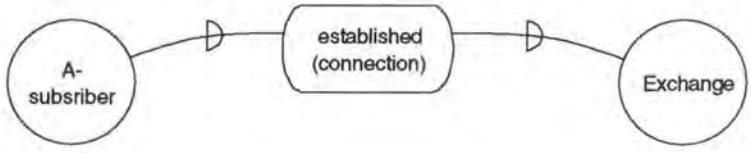


Figure 3.3 : a goal dependency

- "In a **task dependency**, the depender depends on the dependee to carry out an activity. A task dependency specifies how the task is to be performed, but not why. The depender is vulnerable since the dependee may fail to perform the task (...) Task specifications should be viewed as constraints rather than as the complete (and therefore adequate) knowhow for performing the task."

Figure 3.2. doesn't show any task dependency but one could easily imagine how to introduce that particular kind of dependency between for example the patient and the assurance company. Let's say that after having provided the certificate, the patient has to fill in a form before receiving the repayment. The assurance would then ask the patient answering some questions in a specific way without explaining why. This would be modelled by a task dependency.
In the communication exemple, the A-subscriber knows he/she has to dial the number of the B-subscriber if he/she wants the operator to establish the connection between them (without knowing how the operator will use these digits to connect them).

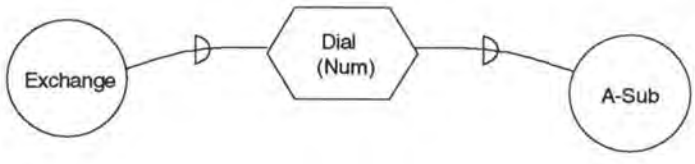


Figure 3.4 : A task dependency

- "In a **resource dependency**, one actor (the depender) depends on the other (the dependee) for the availability of an entity (physical or informational). By establishing this dependency, the depender gains the ability to use this entity as a resource."

The assurance dependency on patient's fee is modelled as a resource dependency as well as the dependency for the delivery of a certificate between the doctor and his/her patient, the communications payment dependency between the A_subscriber and the Exchange, ...

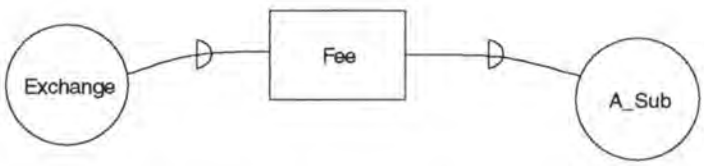


Figure 3.5: A resource dependency

For these three types of dependencies, the dependum is well defined in the sense that both the depender and the dependee know exactly what they are waiting for or what they have to do or to furnish. But it is not always the case. It may happen that the dependum isn't as sharply defined as we would like and this kind of particular dependency will be described by a softgoal dependency.

- "In a **softgoal dependency**, a depender depends on the dependee to perform some task that meets a softgoal. The meaning of the softgoal is specified in terms of the method that are chosen in the course of pursuing the goal. As in a goal dependency, a depender gains the ability of having the goal condition brought about, but becomes vulnerable in case the dependee fails to bring about that condition. The difference here is that the condition to be attained are elaborated as the task is performed."

The patient pays the full price for the medical visit. He knows he/she'll have a repayment via the assurance and he/she'd like to be repayed as quickly as possible (figure 3.2). What is meant by « quickly » is not really defined.
In the communication exemple, the A-subscriber certainly wants the operator to establish a « safe » connection with the B-subscriber. This could be modeled by a softgoal dependency because the notion of safety is not clear-cut.

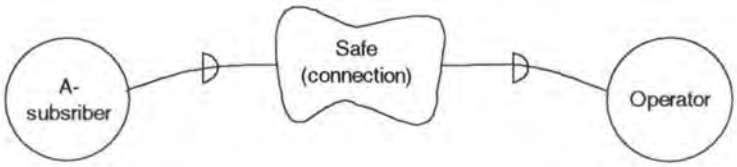


Figure 3.6: A softgoal dependency

Dependency strength

The Istar framework also allows different strength degrees attached to the dependencies. Effectively, the convict to deth prisoner depends contingently on the king for a possible reprieve and he/she also depends on his/her warder for a last cigarette. Obviously, the prisoner won't grant the same importance for the two dependencies.
These different degrees of strength can be applied on both sides of a dependency.
« On the depender side, a stonger dependency means the depender is more vulnerable, and is likely to take stronger measures to mitigate vulnerability. On the dependee side, a stronger dependency implies that the dependee will make a greater effort in trying to deliver the dependum. ».

Istar Provide three different types of dependency forces:

- "In an **open dependency**, a depender would like to have the dependum goal achieved, task performed, or resource available, so that it could be used in some course of action. (...) On the dependee side, an open dependency is a claim by the dependee that it is able to achieve the dependum for some depender."

Graphically, an open dependency will be marked by an "**o**".
If we take again the example of the prisoner, the dependency for the cigarette would be an open dependency on the depender side.

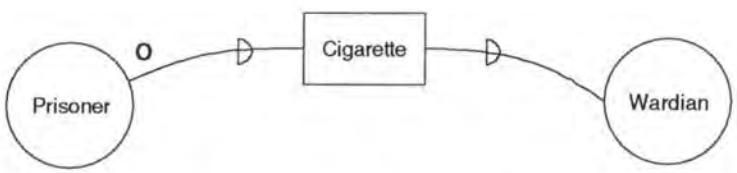


Figure 3.7: Open dependency

- "In a **committed dependency**, the depender has goals which would be significantly affected- in that some planned course of action would failed - if the dependum is not

achieved (...) On the dependee side, a committed dependency means that the dependee will try its best to deliver the dependum."

Graphically, a committed dependency is unmarked.

Figure 3.8: Committed dependency

- "In a **critical dependency**, the depender has goals which would be seriously affected - in that all known courses of action would fail - if the dependum is not achieved. In this case, we assume that the depender would be concerned not only about the viability of this immediate dependency, but also about the viability of the dependee's dependencies, and the dependee's dependee's dependencies, and so forth."

Graphically, we use an "**X**" for a critical dependency.

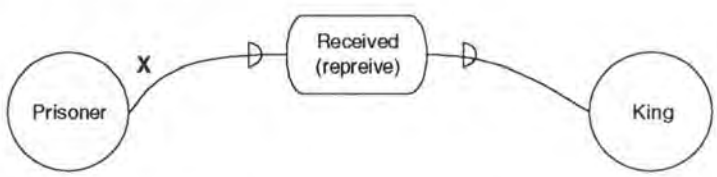


Figure 3.9 : Critical dependency

Agents, roles and positions

One can extend the basic Strategic Dependency model by refining the notion of actor. Agent, Role and Position are three possible specializations of the notion of actor which provide different views of the organization.
"A role is an abstract actor. Dependencies are associated with a role when these dependencies apply regardless of who plays the role [...]
An agent is an actor with concrete, physical manifestations, such as human individual [...]
A position is intermediate in abstraction between a role and an agent."
These three notions are related as follows (see figure 3.10):
An agent occupies a position;
A position covers a role.

Furthermore, these three notions can have subparts and each part or subpart is considered to be intentional.

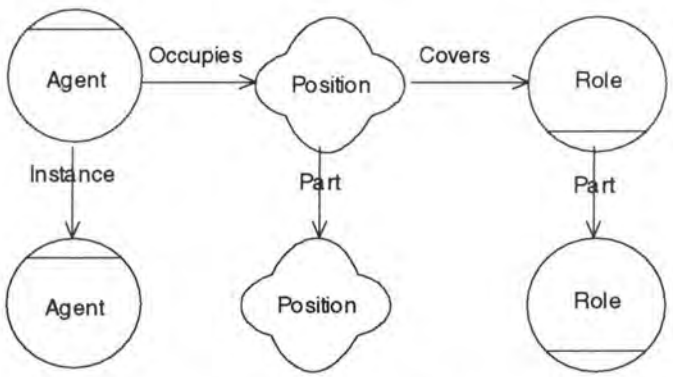


Figure 3.10: Graphical representation of Agent, Role, Position

Figure 3.11 shows a simplified example of a SD model using agent, role and position for the communication between subscribers.

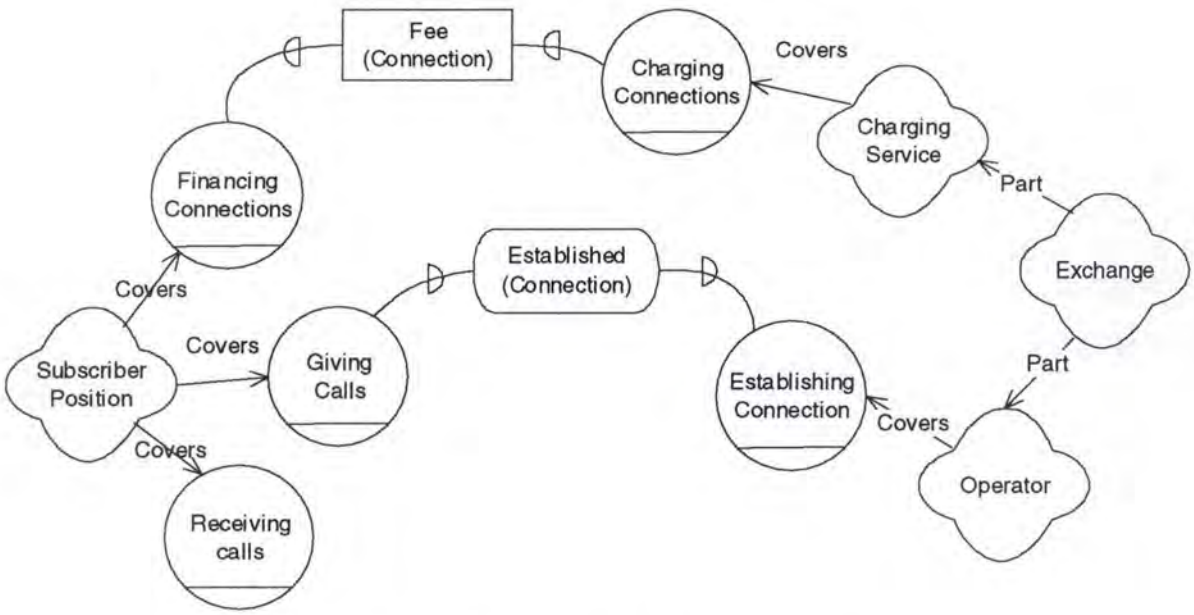


Figure 3.11: Agent-Role-Position for the communication exemple

The Strategic dependency model can help us to achieve a deeper understanding of a work process. It represents the set of intentional dependencies among the actors of the process and by following the model links, one can find out who are the more vulnerable actors and for what.

It's never good seeing that an actor depends on every other actors in a same work process, and this observation usually leads to the search of some ways in order to mitigate this vulnerability. Three mechanisms can contribute to enforce an actor position and thus reduce the risks inherent to this actor's dependencies:

- A depender would be less vulnerable if in his turn, he had some influence on the dependee. "A commitment is **enforceable** if there is some way for the depender to cause some goal of the dependee to fail, e.g., if there is a reciprocal dependency."
  Imagine a brewery, it will depend on a cafe for the sale of its beer, but reciproquelly, the bistro will depend on the brewery for stocking up with beer and thus beeing able to serve its clients.

- A second possibility is the **assurance** mechanism and is defined in these terms: "Assurance means that there is some evidence that the dependee will deliver the dependum, apart from the dependee's claim."

- Finally, "**Insurance** mechanisms reduce the vulnerability of a depender by reducing the degree of dependence on a particular dependee."
  Let's take again the brewery example. We said that the brewery depended on a bistro to sell its beer but usually, the brewery is "associated" with more than one cafe. That means that if one of them fails to perform its task consisting in selling the beer, the brewery won't be affected too badly.

These different mechanisms should balance the dependencies and thus enforce the whole work process by enforcing the position of each actor.

3.3. The Strategic Rationale Model

In the previous section, we saw how the Strategic Dependency model of the Istar framework could underline the intentional dependencies among actors of a work process. However, this model stays very general in the sense that only the external dependencies will be modelled. The internal dependencies within an actor remain hidden.
The Strategic Rationale model will help to understand the internal structure of an actor therefore, we will be able to describe and support this actor's reasoning.

3.3.1. Modelling features

The Strategic rationale model can be described as a graph composed of nodes and links.
There exist four types of nodes already explained in section 3.2 (the goal, the task, the resource and the softgoal), and two types of links (the means-ends and the Task-decomposition links).
The purpose of this graph is to model the actor's "ways of doing things" in order to achieve a particular goal, task or softgoal. One can say that the SR model is strategic in the sense that we'll only model the elements considered important enough in the achievement of a particular goal.

The **task-decomposition links** are used to link a task to its sub-components. There exist four different types of task-decomposition links based on the sub-component of a task: subtask, subgoal, resourceFor and softgoalFor.
Figure 3.12, one can see that the main goal of the assurance company is the covering of its members, and it is represented by the task (*Cover member*). The assurance will thus ask a fee to each member (*Takes a fee*) and when it is needed, it will give them a repayment (*repays*).
The two task-decomposition links used to decompose (*cover member*) are two subtask.

What is interesting and new in this model is the freedom let to the actor at each level of the decomposition. There can be several ways to achieve a particular goal and this will be modelled by the **means-ends links** between the main goal (the end) and the ways (the means) for achieving this goal. Graphically, a means-ends link is represented by an arrow going from the sub-component (the means) and pointing towards the principal goal (the end).
The assurance has to repay some of its members and it has the choice between two different ways to do this. Either it repays the patient directly, or it repays the patient's doctor who asked the reduced price to his/her patient (see figure 3.12).

When an actor has different choices or different ways in achieving some particular goal, we introduce the notion of **routine**.

"A routine is a subgraph in the SR graph with a single link to a 'means' node from each "end" node [...] the notion of a routine is used to refer to one process and its rationales "

Figure 3.12, one can find out two different routines. On one hand, we have the subgraph including "repays patient" and the subgraph that includes "repays doctor".

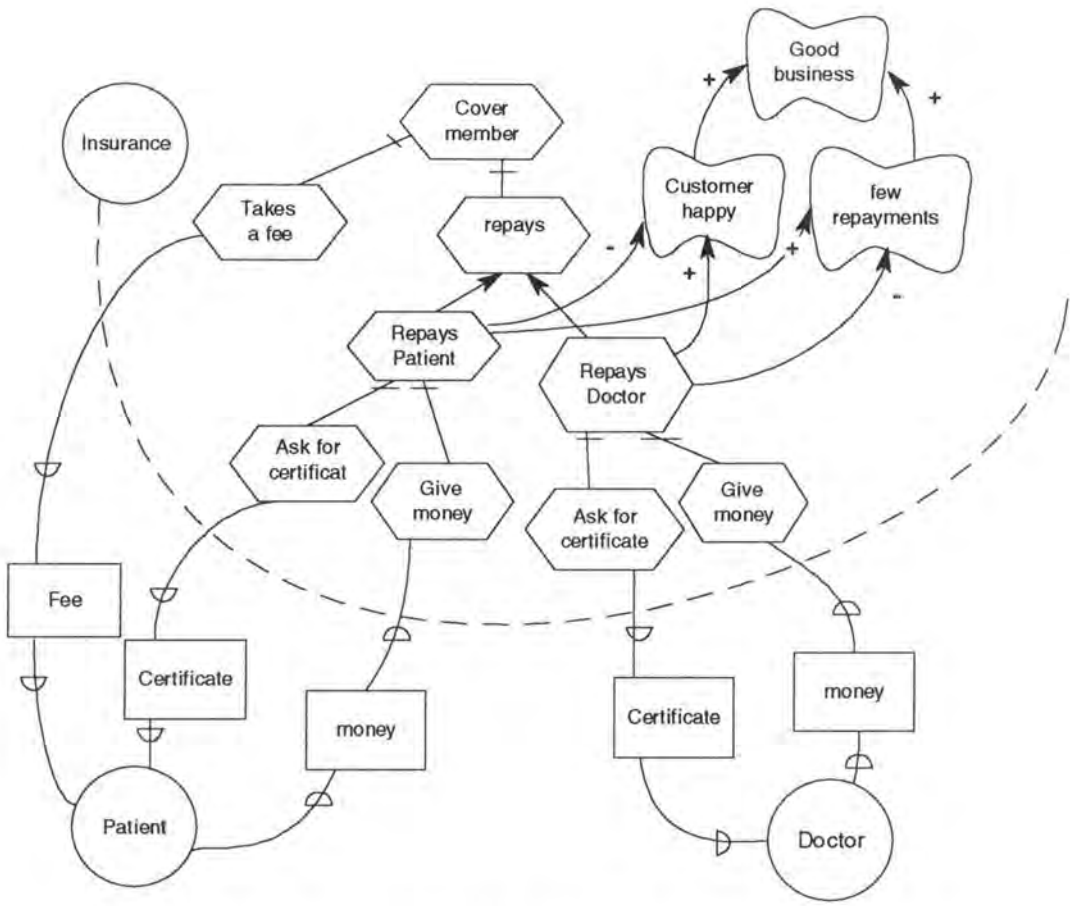


Figure 3.12: An SR Model showing alternative ways of accomplishing 'Repays'

In the SR Model, the softgoals will be handeled differently. One will rather see them as qualitative attributes and each "ways of doing things" or routine, will affect these non-functional goals positively or negatively. In a same way, a qualitative goal can also affect or influence another qualitative goal.

For example, figure 3.12, making the customer happy constitues a qualitative goal for the assurance company. If the assurance decides to repay directly the doctor instead of its member, this choice will contribute positively in making the customer happy. If the customer is happy, it will contribute to make the business prosperous. The assurance company will also make a good business if it doesn't have to make too many repayments and one can argue that if the

assurance repays the doctor instead of the member, it will contribute to increase its repayments. Indeed, the patient won't have to handle certificates anymore, he/she won't have too go to the assurance office for the repayments; this could encourage them to go more often to see a doctor and therefore, this could lead to increasing the repayments to the doctors.

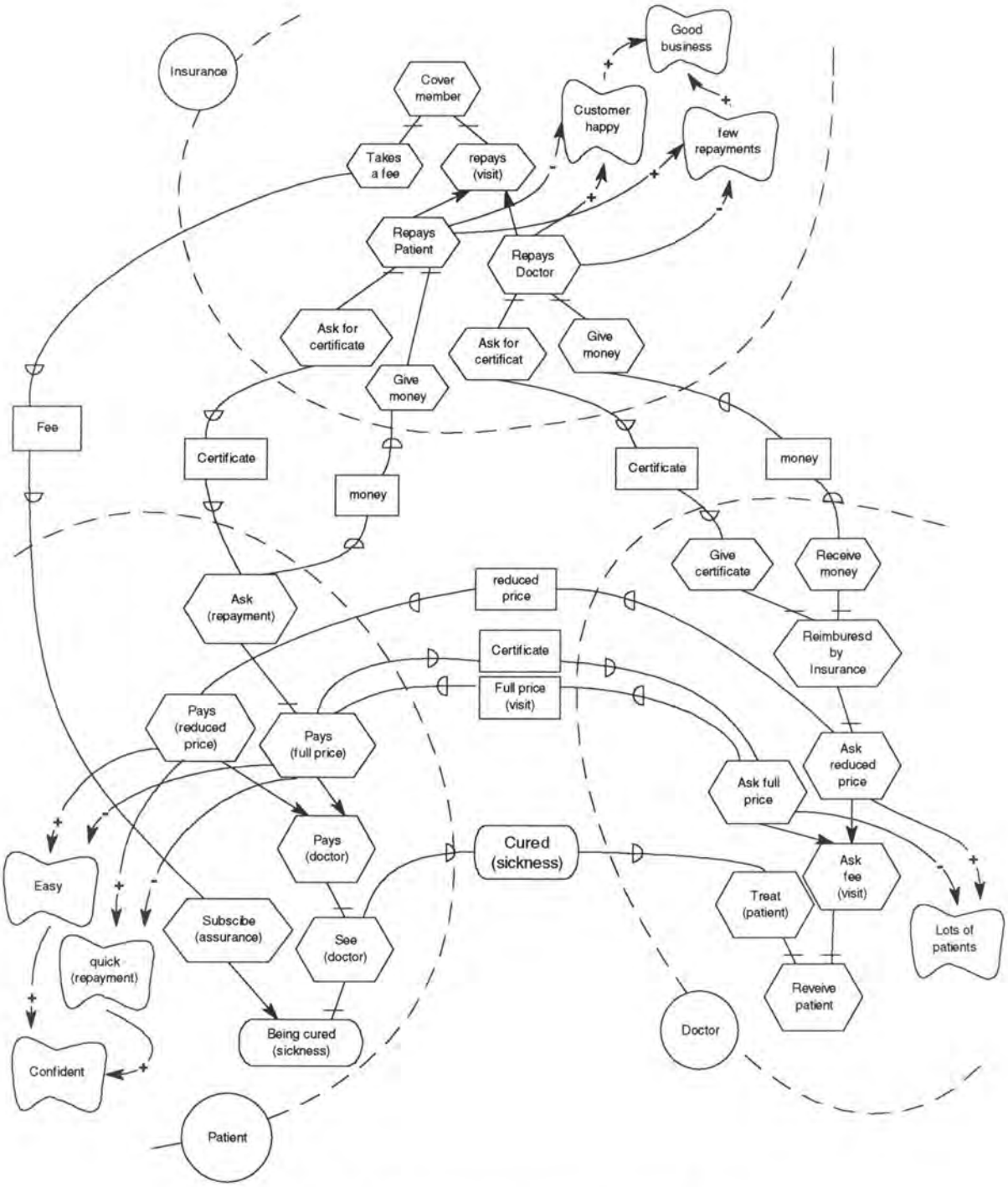


Figure 3.13: SR Model for the assurance company process

In a SR model, it may also happen that a reasoning includes dependencies on external actors. In such cases, we'll use the dependency links already explained in the Strategic Dependency model.

In the routine including (*repays patient*), the assurance actor depends on its members for the certificates in order to be able to repay them. We have represented this by a resource dependency going from the inside of the assurance boundery towards the Patient.

In order to have a complete view of the process, it is possible to relate the different SR models of the main actors. One can thus analyze the qualitative goals of each stakeholder and understand why they privilagiate one way of working instead of another and how a particular routine can alter the behaviour of another actor (see fig 3.13)

Chapter 4: Formal specification of a library system

4.1. Introduction

Chapters 2 and 3 have proposed two different frameworks (Albert and i*) used in the requirements engineering and process modelling activities.

In this chapter, we will try to show how these two approaches, following different views, could be related to a same case study, how they can be complementary and how used together, they will lead to a full comprehension of the work process. The case study will be a library system, managing book acquisition and loan policies.

We'll then try to answer the question of knowing if a systematical binding could be established bewteen Albert and i*, in order to induce an SD or SR model from Albert formal specifications and vice versa.

4.2. Informal description of the case study

We will now describe the academic case study related to the management of a library [Darimont].
In this section, we will first present an informal specification of the library main functionalities.

4.2.1. Definition

The library system aims at supporting the management of a library that is, managing the book stock and the book loans.
There are three classes of users:

- the staff members;
- the ordinary members;
- the extra-members;

and two classes of books:

- the critical books which may not be borrowed by ordinary and extra-members;
- the borrowable books.

4.2.2. Support

- two catalogues refer to all books of the library. One of them is indexed by the book authors, and the other, by keywords associated with the books;
- a list of the book loans for ordinary and extra-members (the staff members who want to borrow a book don't have to mention it to the Librarian, they just have to choose the book and to bring it back once they have read it);
- A list of the different book orders in progress;
- A complete list of the registered members of the library.

4.2.3. Access to the library

Staff members can acces the library at any time. The library can be entered by ordinary and extra members only during the opening hours. Moreover, the extra-members must be explicitely allowed to get to the library by a staff member.
Any library member must have been registered by the Librarian, before his first access .
The Librarian will then have to keep the list of registered members up to date.

4.2.4. Book acquisition policy

The staff members are responsible for the good quality of the library contents. They decide which book to order, where to classify a new book, which book may not be borrowed by the ordinary users and which book must be removed from the shelves.
As we just mentioned it, books are ordered upon request of the staff members. They transmit the book identification to the librarian who will compose the order. An order can only be issued if the book identification is complete and if there is a staff member (not necessarily the one who initiated the book ordering) who agrees to pay the bill.
The librarian sends the order either to the cheapest bookshop or to the one which will be able to satisfy the order the most quickly, depending on the priority set by the staff member who ordered the book.
When a book is received, a staff member provides the category and the list of keywords associated with the book. The publication year and the book number are determined by the librarian. The Staff members put the new books in the library shelves by increasing order of reference.

When a librarian introduces a new book, he updates the catalogues by inserting a new entry for each author and for each associated keywords list. An entry in a catalogue contains three parts:
- the index;
- the complete book identification;
- a book reference.

A book reference is a triplet:
- a category which belongs to a predefined list;
- the year of the book publication;
- a number which identifies a book univocally for a given category and year. A letter is appended to the triplet in order to distinguish duplicates.

4.2.5. Loan policy

Books may be borrowed freely by staff members while ordinary and extra-members have to pay a small fee. The users of the library remove directly from the shelves the books they want to borrow.
The ordinary users may keep maximum two borrowable books at once and for 15 days only. Nevertheless, a loan period for an ordinary member can be extended once for 15 days if nobody else is waiting for the book. The period of time during which an extra-member may keep a book is fixed by the staff member who allowed the extra-member to borrow the book.

Actually, delay to returning a book is only detected when another request for the borrowed book occurs. In these circumstances, the librarian either phones the indelicate member and prompts him to return his/her book copy as soon as possible, or sends him a reminder (letter or electronic mail).

When a member returns a borrowed book, the Librarian puts it aside; after a while, a Staff Member takes all the returned books and puts them back on the library shelves. After having been returned, a book have to be replaced on its shelf within one day.

If an ordinary or extra member becomes too undisciplined, a staff member will be able to forbid the access of the library.

4.3. Specification before any change

4.3.1. Albert specification

a) Agents identification

From the informal description of the Library System given in section 4.2., we can find out six terminal agents.
If we adopt a functional view of the system, the agents identification could be the following: inside the Library System, we have the Extra Member agents and the Ordinary Member agents. They all are users of the Library.
We also have the Bookshop agents, playing a role in the acquisition policy of the library.
Inside the Library society itself, we have a Librarian responsible for the whole management of the library, and Staff Member agents who help the Librarian for the management. We also introduce an agent representing the bookstore available for the members of the library.

The graphical representation of the Library System is depicted figure 4.1.

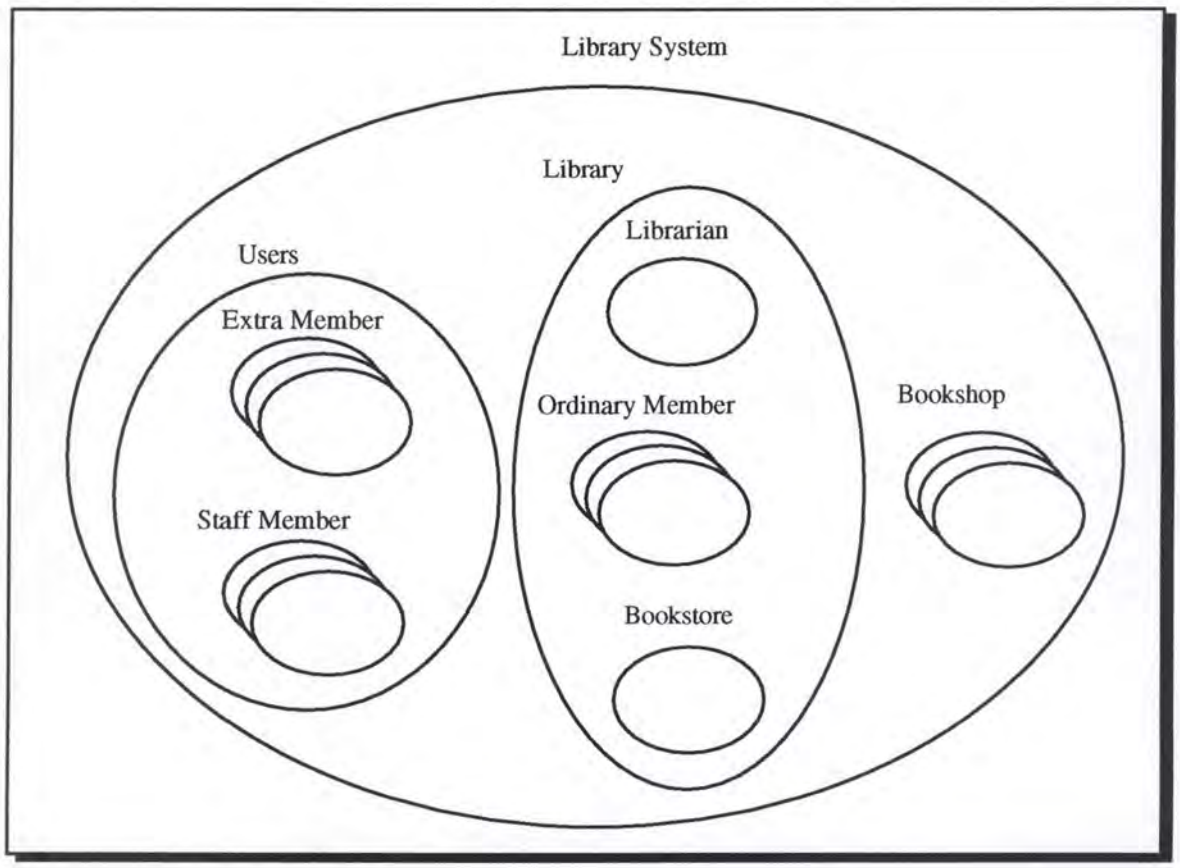


Figure 4.1 : Graphical declaration of the Library system society

b) Book acquisition policy

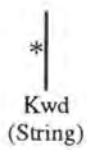
Declaration of the Librarian agent

The graphical declaration of the Librarian agent is depicted on figure 4.2. One can see that the state structure is composed of:

1. Six internal state components (Cat_Book_KW, Cat_Book_Author, Table_Member, B_To_Classify , Ord_List and the individual element Todays_Date).

Cat_Book_KW: This state component represents a catalogue refering to all books of the Library. it is indexed by a list of keywords (*KW_List*) associated with a list of books (*L_Book*).

- The type *KW_List* can be defined as:



The index is characterized by a set of keywords of type *String*.

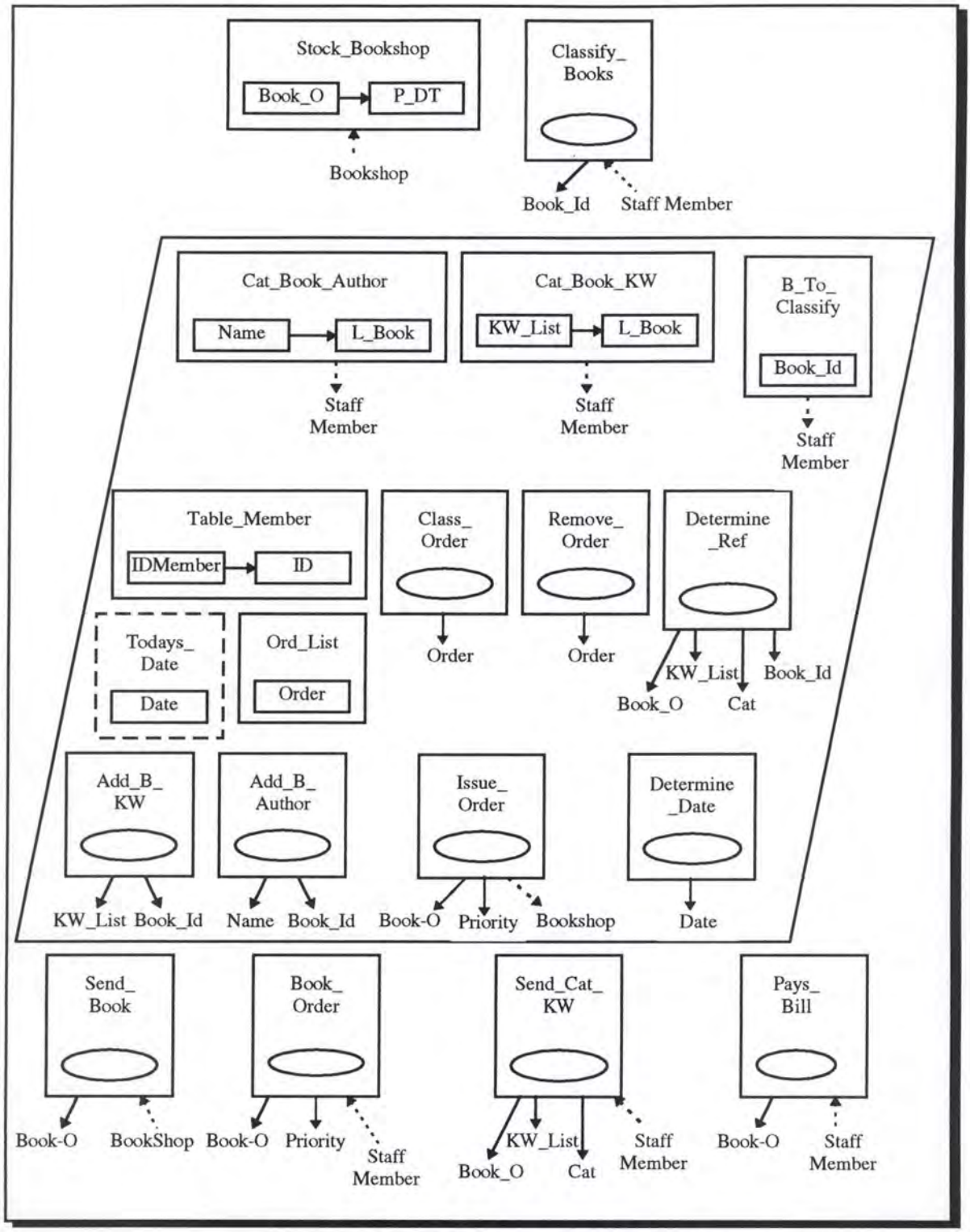
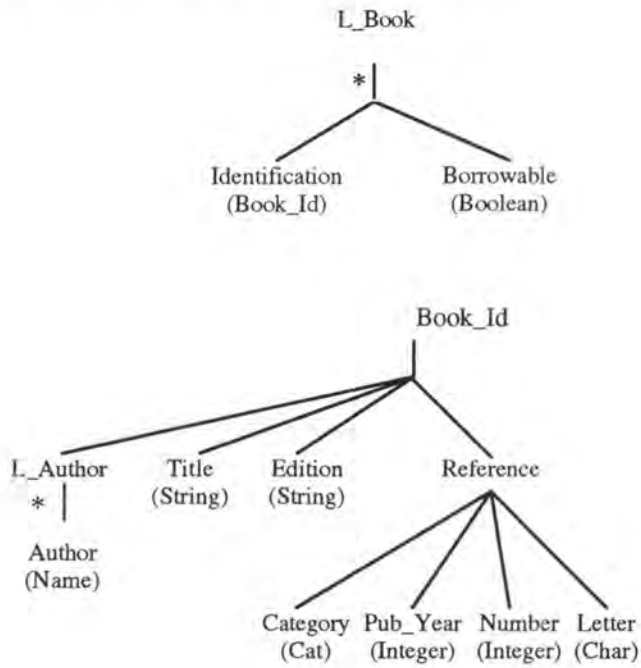


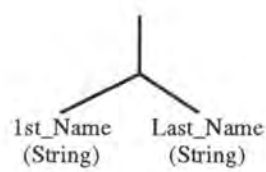
Figure 4.2 : Graphical declaration of the Librarian agent

- The types L_*Book* and *Book_Id* can be defined as:



A Book in a catalogue is characterized by its complete identification (its author(s), title, edition and a reference composed of a category which belongs to a predifined list, the year of the book publication, a number which identifies a book for a given category and year and a letter in order to distinguish duplicates), and by a status telling if the book is borrowable or critical.

- The type *Name* can be described as:



An author is described by its first and last names.

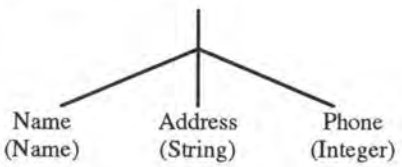
Cat_Book_Author: This component also represents a catalogue refering to all books of the library. It is indexed by the authors (Name) associated with a list of books (L_Book).
The two types have already been described.

Table_Member: this table contains all the members registered in the library. It is indexed by the category of a member (IDMember) associated with the complete identification of the member (Member).

- The type *IDMember* can be defined as:

A Member registered in the library is either a staff member or an ordinary member or an extra member. The types represent the type of identifiers associated with agent classes.

- The type *Member* is the following:

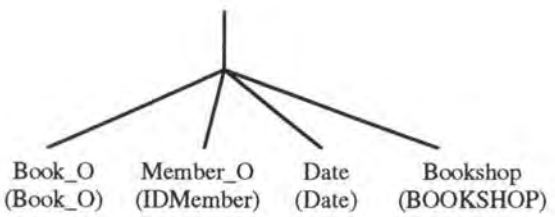


A member is characterized by his/her name, address and phone number.

B_To_Classify: it represents a set of books (Book_Id) which wait to be classified in the library Bookstore by a Staff Member.

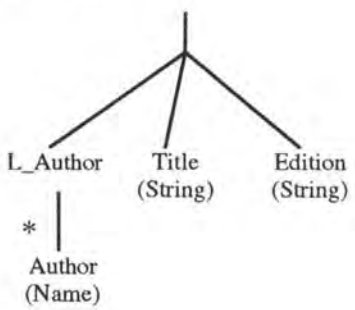
Ord_List: This list represents all the orders (*Order*) in progress.

- The type *order* can be defined as:



An order is defined by the identity of the book ordered, the name of the staff member who ordered the book, the date and the identity of the bookshop where the order has been issued.

- The type *Book_O* is the following:



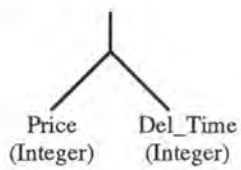
A Book is identified by its author(s), title and its edition.

Todays_date: this individual entity represents the date (*Date*) of the current day.

2. The Librarian agent has also one external state component (Stock_Bookshop).

Stock_Bookshop: This table is from the Bookshop initiative. It represents for each book (Book_O) present in a Bookshop stock, its price and its delivery time (P_DT).

- therefore, the type P_DT can be represented as:



3. The state structure of the Librarian agent is also characterized by seven internal actions (Determine_Date, Issue_Order, Class_Order, Remove_Order, Add_B_KW, Add_B_Author, and Determine_Ref).
Determine_Date: the Librarian determines the current date (Date)

Issue_Order: the Librarian agent sends a book order (Book_O) to a bookshop at a certain priority.

Class_Order: he/she classifies a new order (Order) in progress in the Ord_List.

Remove_Order: when the librarian has received an ordered book, he/she can remove the order (Order) from the Ord_List.

Add_B_KW: he/she adds a new book (Book_Id) in the Cat_Book_KW catalogue with a list of keywords (KW_List) as index.

Add_B_Author: the Librarian adds a new book (Book_Id) in the Cat_Book_Author catalogue with the name (Name) of the author as index.

Determine_Ref: the librarian determines the reference of a book (Book_O) with its category (Cat) and its Keywords list (KW_List).

4. The Librarian perceives also five external actions (Classify_Books, Send_Book, Book_Order, Send_Cat_KW and Pays_Bill)

Classify_Book: a Staff member decides to classify in the library Bookstore, a book (Book_Id) contained in the B_To_Classify list.

Send_Book: a bookshop has sent, an ordered book (Book_O) to the librarian.

Book_Order: a Staff Member asks the Librarian to issue an order for a book (Book_O) to a Bookshop at a given priority (Priority). The priority will either be a priority in time or in price.

Priority = {Time, Price}

Send_Cat_KW: a Staff Member provides the librarian with a category (Cat) and a list of keywords (KW_List) for a given book (Book_O).

Pays_Bill: a staff Member pays the bill for an ordered book (Book_O).

Constraints

LIBRARIAN

BASIC CONSTRAINTS

- **Initial Valuation**

B_To_Classify = { }
* *There are no books to classify in the library Bookstore*

Ord_List = { }
* *There are no orders in progress.*

LOCAL CONSTRAINTS

- **State Behaviour**

in (B_To_Classify, b) $\Rightarrow_{>1day} \neg$ in (B_To_Classify, b)
* A *book must be classified in the library Bookstore within one day.*

- **Effect of Actions**

Determine_Date(d): Todays_Date = Date
** the librarian sets the current date*

Determine_Ref(bo, kwl, c, b): B_To_Classify = add(B_To_Classify, b)
** When the librarian has determine the reference of the new book, he/she puts it in the B_To_Classify list*

Remove_Order(O): Ord_List = Remove(Ord_List,O)
** An order about a book (b) is removed from the list 'Ord_List'.*

Class_Order (O): Ord_List = Add (Ord_List,O)
** A new order in progress, is registered in the orders list.*

Add_B_Author (b): Cat_Book_Author = Add (Cat_Book_Author[n], b)
with n ∈ L_Author(b)
** A new book is registered in the author catalogue.*

Add_B_KW (kw, b): Cat_Book_KW = Add (Cat_Book_KW[kw], b)
** A new book is registered in the keyword catalogue.*

Staff Member.Classify_Books (b): B_To_Classify = remove (B_To_Classify ,b)
** When a Staff Member classifies a book in the library, he/she removes it from the list "B_To_Classify".*

- **Causality**

Staff Member.Book_Order (b, pr) ; _ . Pays_Bill (b) → Issue_Order (b, pr).bs;
Class_Order(O)
with Book_O(O) = b
Member(O) = Staff Member
Date(O) = Todays_Date
Bookshop(O) = bs
** When a Staff Member asks the librarian to order a book(b) at a given priority and if a Staff Member agrees to pay the bill for that book, then the librarian will issue an order to the appropriate bookshop and will register the order in the orders list.*

Staff Member.Send_cat_KW(b, kw, c) → Determine_Ref (b, c, book)

with Authors (book) = Authors (b)
∧ Title (book) = Title (b)
∧ Edition (book) = Edition (b)
∧ Category (book) = c;
Add_B_KW (kw, book);
Add_B_Author (book)

** When a staff member sends the category and a list of keywords associated with a book, the librarian can determine the reference of that book and can register it in the two library books catalogues*

Bookshop.Send_Book (b) → Remove_Order (O)
with Book_O(O) = b

** When a bookshop has sent a book, the librarian removes one order he chooses from the orders list which asked for that book*

- **Capability**

F (Issue_Order(b, p).bs / (Isof p = "price" ∧
∃ bs' : Price(bs'.Stock_Bookshop[b])
< Price(bs.Stock_Bookshop[b]))
∨ (Isof p = "Del_Time" ∧
∃ bs' : Del_Time(bs'.Stock_Bookshop[b])
< Del_Time(bs.Stock_Bookshop[b]))

** the librarian can't send an order to a bookshop who doesn't respect the priority in time or in delivery time*

COOPERATION CONSTRAINTS

- **Action Perception**

I (Bookshop.Send_Book(b) / ¬ in (Ord_List, O): book_O (O) = b
∧ bookshop (O) = bookshop)

** the librarian ignores the book sent by a bookshop, which hasn't been ordered*

I (Staff member.Classify_Books (b) / ¬ in (B_To_Classify, b))

** the librarian ignores the action of a staff member consisting in removing a book that doesn't exist in the "B_To_Classify" list*

- **State Perception**

$\mathcal{K}_K$ (b.Stock_Bookshop[-]/TRUE)

** the librarian can always perceive the catalogues of the bookshops*

- **State Information**

$\mathcal{K}_K$ (Cat_Book_Author.Staff Member/TRUE)

$\mathcal{K}_K$ (Cat_Book_KW.Staff Member/TRUE)

$\mathcal{K}_K$ (B_To_Classify.Staff Member/TRUE)

* the "Cat_Book_Author", "Cat_Book_KW" catalogues and the "B_To_Classify" list are always visible to the staff members.

- **Action Information**

$\mathcal{K}_K$ (Issue_Order(b, p).Bookshop/TRUE)

Declaration of the Staff Member agent

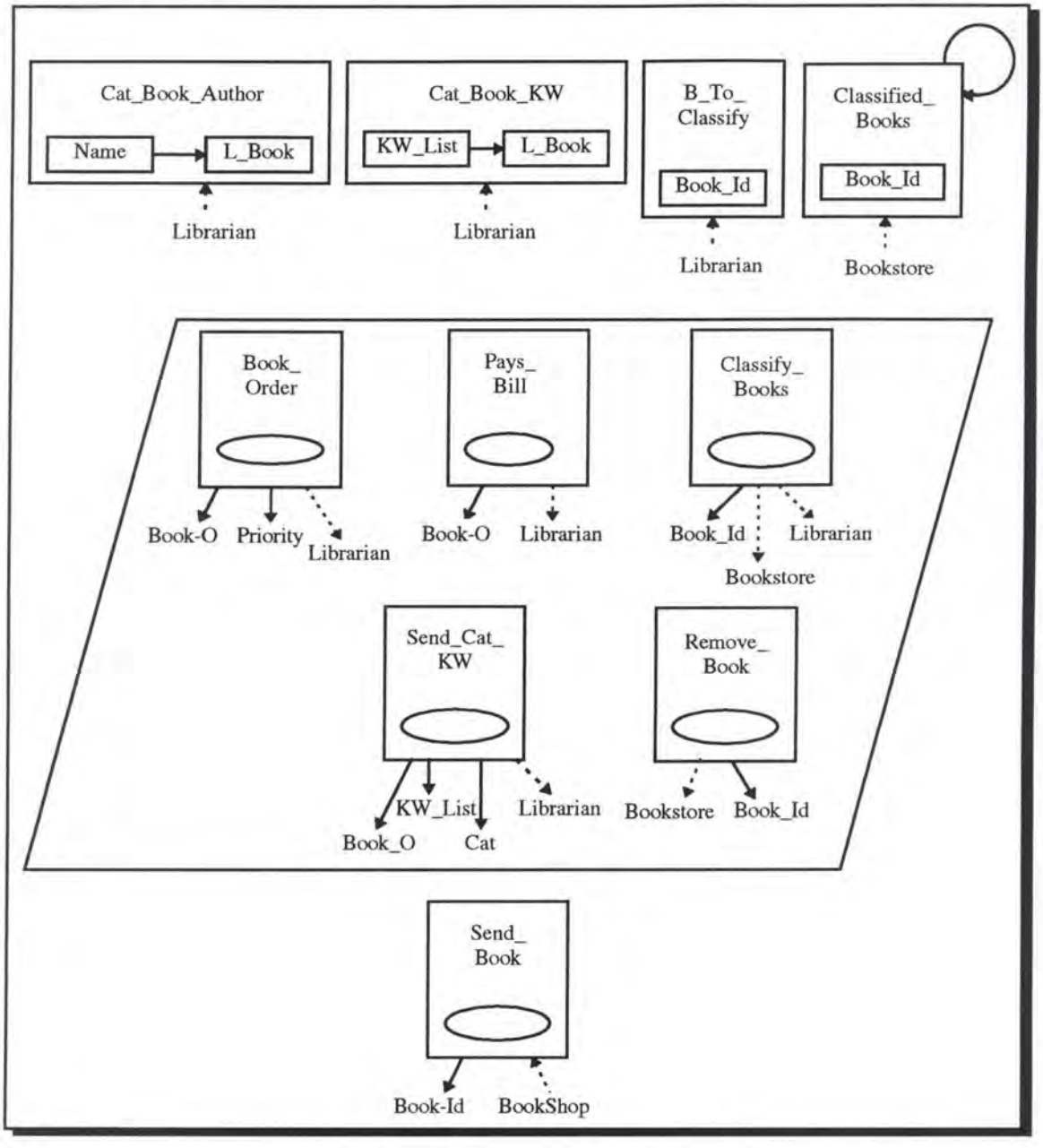


Figure 4.3 : Graphical declaration of the Staff Member agent

The graphical declaration of the Staff member agent is described on figure 4.3. We'll only describe the elements which havn't been already described in the previous declaration.

1. The Staff member agent has four external state components (Cat_Book_Author, Cat_Book_KW, B_To_Classify and Classified_Books)

Classified_Books: it represents the sequence of the books (Book_Id) present on the library Bookstore.

2. We can see five internal actions (Book_Order, Pays_Bill, Remove_Book, Send_Cat_KW and Classify_Books).

Remove_Book: this action represents the fact that a Staff Member removes a given book (Book_Id) from the library Bookstore.

Classify_Book: a Staff Member classifies a book (Book_Id) in the library Bookstore.

3. Finally, this agent perceives one external action (Send_Book) already described.

Constraints

STAFF MEMBER

LOCAL CONSTRAINTS

- **Causality**

Bookshop.Send_Book (b)→ Send_Cat_KW (b, kw, c)
** once a bookshop has sent a new book, a staff member sends to the librarian, the category and a list of keywords associated with that book*

- **Capability**

$\mathcal{XO}$ (Classify_Books(b) / ¬ in(B_To_Classify, b))
** a staff member can't classify a book which isn't in the "B_To_Classify" list*

$\mathcal{XO}$ (Remove_Book(b) / ¬ in(Classified_Books, b)
** a staff member can't remove a book which is not in the bookstore*

COOPERATION CONSTRAINTS

- **Action Perception**

$\mathcal{X}$K (b.Send_Book (b) /TRUE)

- **State Perception**

$\mathcal{X}$K (Bookstore.Classified_Books/TRUE)
$\mathcal{X}$K (Librarian.Cat_Book_Author/TRUE)

ℵK (Librarian.Cat_Book_KW/TRUE)

ℵK (Librarian.B_To_Classify/TRUE)

** the staff members always perceive the catalogues, the bookstore and the B_To_Classify lists*

- **Action Information**

ℵK (Classify_Books(b).m /TRUE)

ℵK (Remove_Book(b).b/TRUE)

Declaration of the Bookshop agent

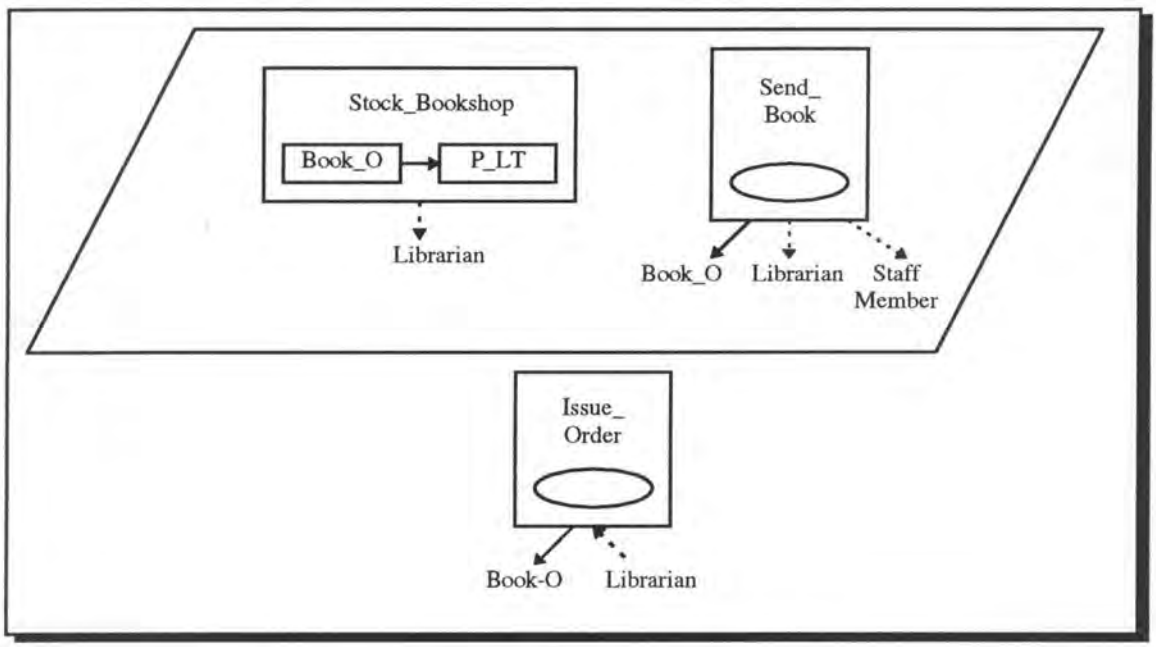


Figure 4.4 : Graphical declaration of the BookShop agent

The different elements of the Bookshop state structure have already been described.

Constraints

BOOKSHOP

LOCAL CONSTRAINTS

- **Causality**

Librarian.Issue_Order (b) → Send_Book (b)

COOPERATION CONSTRAINTS

- **Action Perception**

ℵK (Librarian.Issue_Order (b)/TRUE)

- **Action Information**

ℵK (Send_Book.Librarian (b)/TRUE)
ℵK (Send_Book.Staff Member (b)/TRUE)

- **State Information**

ℵK (Stock_Bookshop.Librarian/TRUE)

Declaration of the Bookstore agent

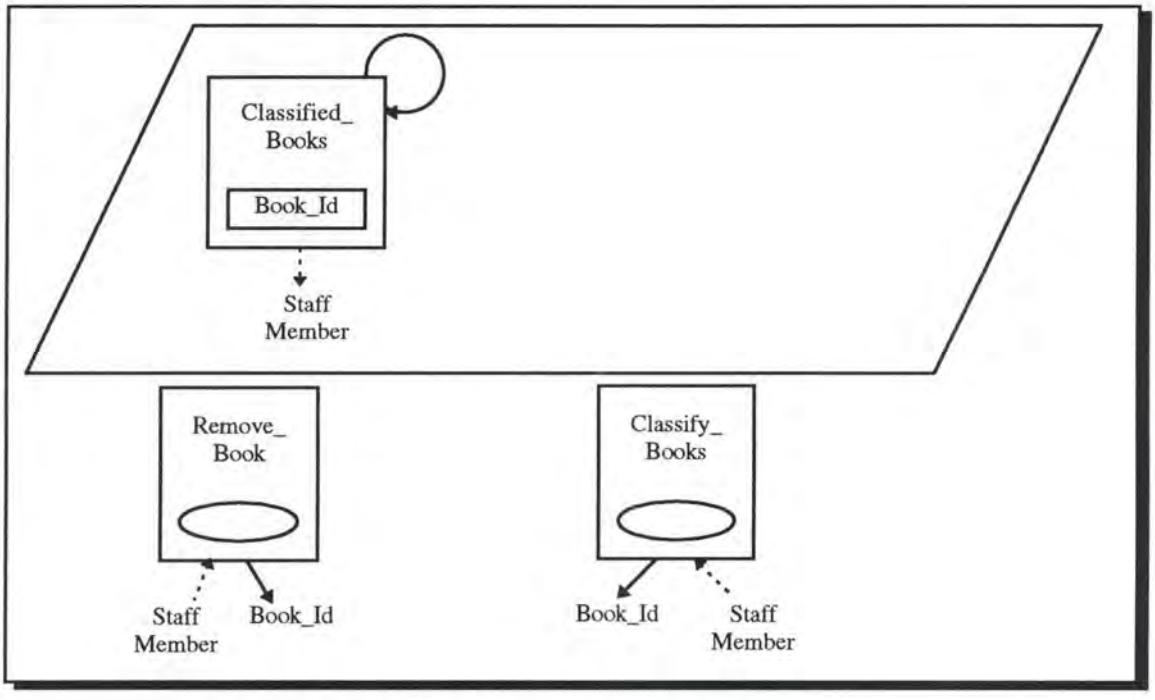


Figure 4.5 : Graphical declaration of the Bookstore agent

All elements of the Bookstore structure have been explained

Constraints

BOOKSTORE

LOCAL CONSTRAINTS

- **Effects of action**

Staff Member.Classify_Books(b): Classified_Books = Add(Classified_Books, b)
Staff Member.Remove_Book(b): Classified_Books = Remove(Classified_Books, b)

COOPERATION CONSTRAINTS

- **Action Perception**

XK (Remove_Book(b).Staff Member/TRUE)
XK (Classify_Books(b).Staff Member/TRUE)

- **State Information**

$\mathcal{X}$K (Classified_Books.Staff Member/TRUE)

c) Loan policy

Declaration of the Librarian agent

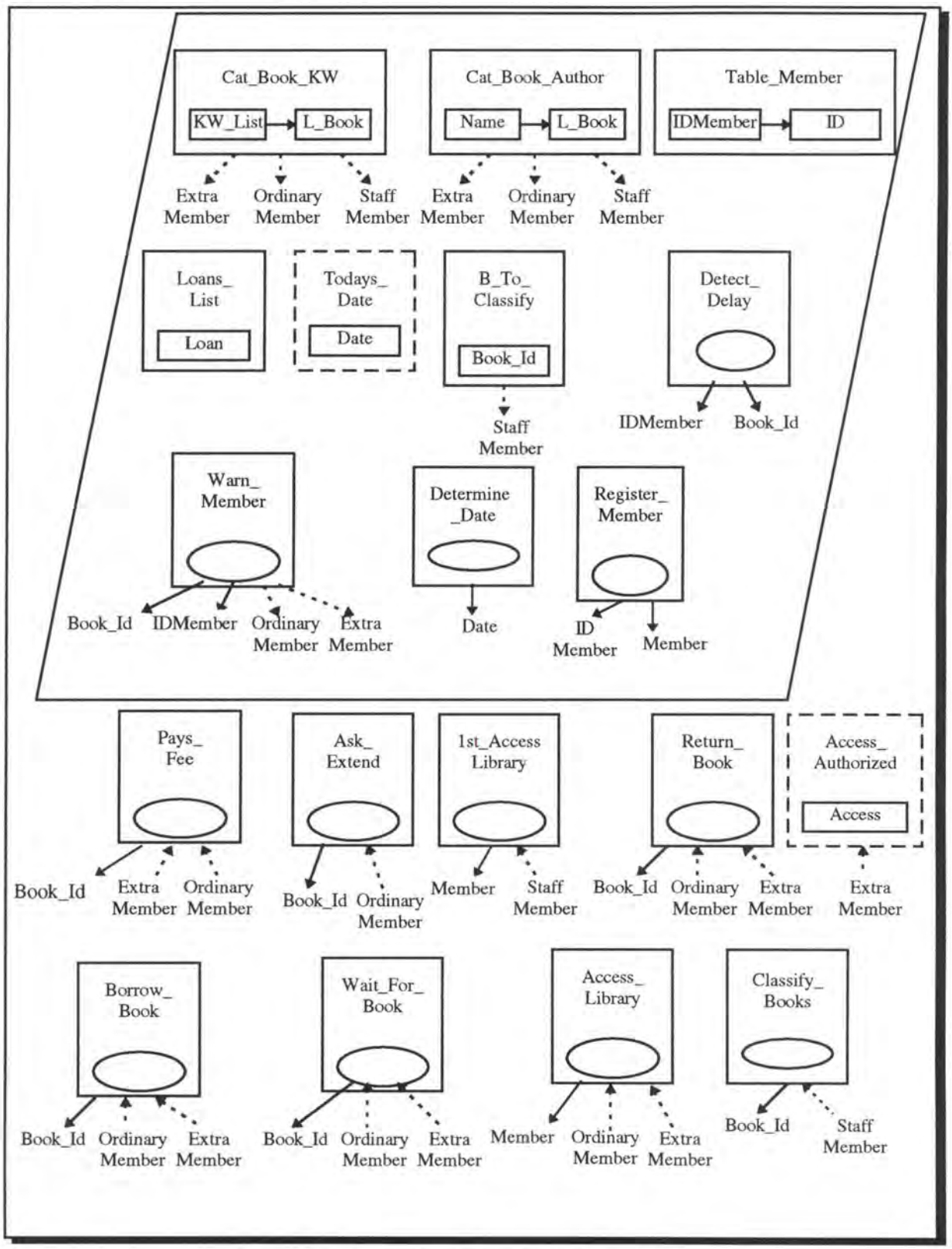


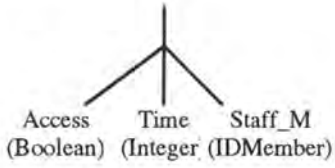
Figure 4.6 : Graphical declaration of the Librarian agent

Figure 4.6 shows us the graphical declaration of the Librarian agent for the book loan process. One can read that:

1. the librarian agent has one external individual state component (Access_Authorized)

Access_Authorized: It indicates the information about the access (Access) of the Extra Members

- The type *Access* is defined by:

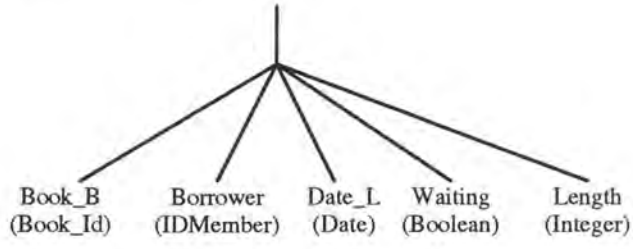


The state component Access Authorized indicates if an Extra Member has access to the library, his/her accorded loan period and the identity of the Staff Member how allowed the library access.

2. this agent has six internal state components (Cat_Book_KW, Cat_Book_Author, Table_Member, Loans_list, B_To_Classify and Todays_Date). Only the Loans_List element hasn't already been described.

Loans_List: this element represents the set of the loans (*Loan*) in process in the library.

- The type *Loan* is described by:



A loan is described by the identification of the book borrowed, by the identification of the borrower, the loan date, an indicator telling if the book is effectively borrowed or if the borrower is waiting after it and by the length of the loan.

3. The Librarian agent has four internal actions (Detect_Delay, Register_Member, Warn_Member, Determine_Date).

Detect_Delay: the Librarian agent detects that a borrower (*IDMember*) is late in returning a given book (*Book_Id*).

Register_Member: this action consists in the registration in Table_Member of a new user (*IDMember*).

Warn_Member: the Librarian warns an Ordinary or Extra member (*IDMember*) of the library for a delay in returning a given book (*Book_Id*).

4. The Librarian perceives eight external actions (Pays_Fee, Borrow_Book, Return_Book, 1st_Access_Library, Ask_Extend, Wait_For_Book, Access_Library and Classify_Books).

Pays_Fee: an Ordinary or Extra member pays a fee for the loan of a book (Book_Id).

Borrow_Book: a member asks the Librarian to borrow a given book (*Book_Id*) that he/she already removed from the library Bookstore.

Return_Book: a member returns a book (*Book_Id*) to the librarian.

1st_Access_Library: a Staff Member indicates his/her first library access to the Librarian in order to be registered.

Ask_Extend: an Ordinary Member asks the Librarian to extend his/her loan length for a given book (*Book_Id*).

Wait_For_Book: an Extra or Ordinary Member asks the Librarian to be on a waiting list for a given book (*Book_Id*) which has already been borrowed.

Access_Library: the Librarian controls all the library accesses of the Ordinary and Extra Members.

Classify_Books: this action has been described in the acquisition policy.

LIBRARIAN

BASIC CONSTRAINTS

- **Initial valuation**

Card(Loans_List) = { }
Card(B_To_Classify) = { }
** There are no loan in progress and no book to classify*

LOCAL CONSTRAINTS

- **Effects of action**

Register_Member(id, m) : Insert(Table_Member, id, m)
** the librarian registers a new member in the table of members*

m.Return_Book(b): Loans_List = remove(Loans_List, l)
with Book_B(l) = b
Borrower(l) = m
** the librarian removes a loan from the loans list*

m.Return_Book(b): B_To_Classify = add (B_To_Classify, b)
** when a member returns a book to the library, the librarian puts it the "B_To_Classify" list*

m.Borrow_Book(b) with b = Book_B(l): in (Loans_List, l):
Loans_List = modify(Loans_List, l)
with Date_L(l) = Todays_Date
Waiting(l) = FALSE
(Isof m = ORDINARY MEMBER
⇒ length(l) = 15)
(Isof m = EXTRA MEMBER
⇒ length(l) = m.Time(Access_Authorized)
** the librarian registers a book loan for an Extra or Ordinary member who was waiting for that book*

m.Borrow_Book(b):Loans_List = add (Loans_List, l)
with Date_L(l) = Todays_Date
Waiting(l) = FALSE
(Isof m = ORDINARY MEMBER

⇒ length(l) = 15)
(Isof m = EXTRA MEMBER
⇒ length(l) = m.Time(Access_Authorized)

** the librarian registers a book loan for an Extra or Ordinary member*

m.Wait_For_Book(b) : add(Loans_List, l)
with Book_B(l) = b
Borrower(l) = m
Date_L(l) = Todays_Date
Waiting(l) = TRUE
Length(l) = UNDEF

** the librarian indicates that a member is waiting for a book*

m.Ask_Extend(b): modify(loans_List, l)
with Book_B(l) = b
Borrower(l) = m
Date_L(l) = Todays_Date

** the librarian extends the loan length of a loan*

m.Classify_Books(b): remove(B_To_Classify, b)

** when a staff member classifies a book, it is removed from the list "B_To_Classify"*

Determine_Date(d): Todays_Date = d

** the Librarian sets the date of the current day*

- **Causality**

m.Wait_For_Book(b) →Detect_Delay(b, bor); Warn_Member(b, bor)

** When the librarian detects a delay, he/she warns the borrower*

Staff Member.1st_Access_Library(m) → Register_Member (Staff Member, m)

** When a staff member enters the library for the first time, he/she will be registered by the librarian*

m.Access_Library(member) with ¬ in-dom(m, Table_mem(m))
→ Register_Member (m, member)

** If an ordinary member tries to acces the library without being registered, the librarian registers him/her in the "Table_Member"*

- **Capability**

$\mathcal{K}$O(Detect_Delay(m,b) / $\exists$ l $\in$ Loans_List : Book_B(l) = b
Borrower(l) = m
Date_L(l) + length(l) > Todays_Date)

** the librarian only detects a delay for a book if the limit date is over*

COOPERATION CONSTRAINTS

- **Action Perception**

XK(m.Classify_Book(b)/TRUE)
XK(m.1st_Access_Library/TRUE)
XK(m.Access_Library/TRUE)
I(m.Ask_Extend(b)/$\exists$ l $\in$ loans_List: Borrower(l) $\neq$ m
Book_B(l) = b
Waiting(l) = TRUE)

** the librarian ignores a loan extend request for a given book if somebody else is waiting for the book*

- **State Information**

XK(Cat_Book_Author.m /TRUE)
XK(Cat_Book_KW.m /TRUE)
XK(B_To_Classify.m/TRUE)

- **Action Information**

$\mathcal{K}$K(Warn_Member(b, bor).m / bor = m)

** the librarian can only warn a borrower how is late in returning a book b*

- **State Information**

$\mathcal{K}$K(Access_Authorized.m/TRUE)

Declaration of the Staff Member agent

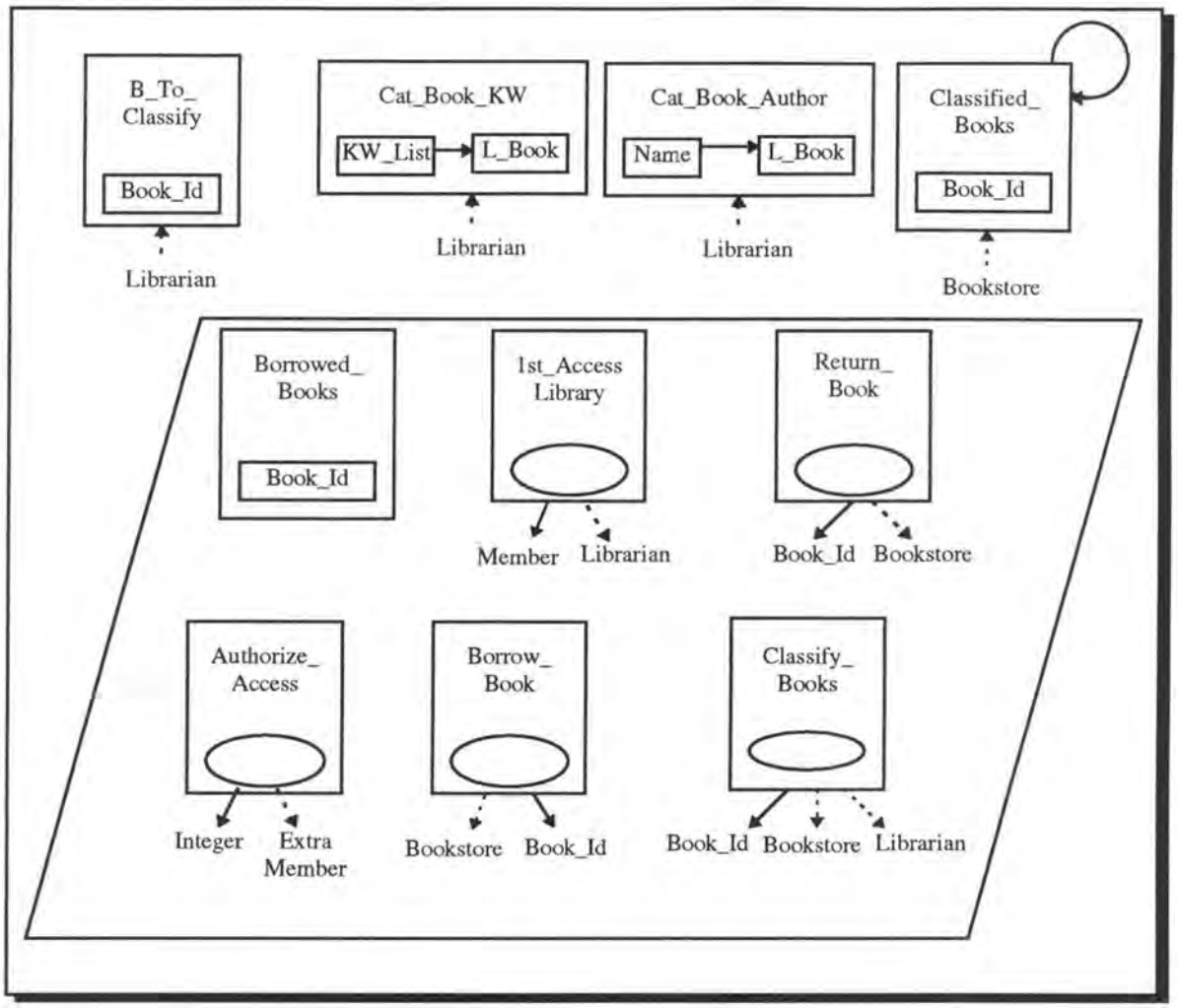


Figure 4.7 : Graphical declaration of the Staff Member agent

The graphical declaration of the Staff Member agent is depicted on figure 4.7. One can see that:

1. This agent has the perception of four external state components (Cat_Book_KW, Cat_Book_Author, Classified_Books and B_To_Classify). These components have been explained before.

2. A Staff Member has an internal state component (Borrowed_Books) representing the set of the books (Book_Id) actually borrowed by this agent.

3. A Staff Member has five internal actions (Borrow_Book, Return_Book, 1st_Access_Library, Authorize_Access and Classify_Books). We will only describe the ones which havn't already been explained.

Borrow_Book: the Staff Member selects a book (*Book_Id*) in the library Bookstore.

Authorize_Access: a Staff Member gives the authorization to an Extra Member to enter the library along with the accorded loan period (Integer).

STAFF MEMBER

BASIC CONSTRAINTS

- **Initial valuation**

Card(Borrowed_Books) = { }
* *the staff member has no borrowed books*

LOCAL CONSTRAINTS

- **Effects of action**

Return_Book(b) : Borrowed_Books = remove(Borrowed_Books, b)
Borrow_Book(b) : Borrowed_Books = add(Borrowed_Books, b)

- **Capability**

XO(Return_Book(b)/in (Borrowed_Books, b))
XO(Classify_Books(b)/in (Librarian.B_To_Classify, b))
XO(Borrow_Book(b)/in (Bookstore.Classified_Books, b))

COOPERATION CONSTRAINTS

- **State perception**

XK(Bookstore.Classified_Books / TRUE)
ℵK(Librarian.Cat_Book_KW / TRUE)
ℵK(Librarian.Cat_Book_Author / TRUE)
ℵK(Librarian.B_To_Classify / TRUE)

- **Action information**

ℵK(Borrow_Book(b).Bookstore / TRUE)
ℵK(Return_Book(b).Bookstore / TRUE)

$\mathcal{K}$K(1st_Access_Library.Librarian / TRUE)
$\mathcal{K}$K(Authorize_Access(i).Em / TRUE)
$\mathcal{K}$K(Classify_Books(b).m / TRUE)

Declaration of the Ordinary Member agent

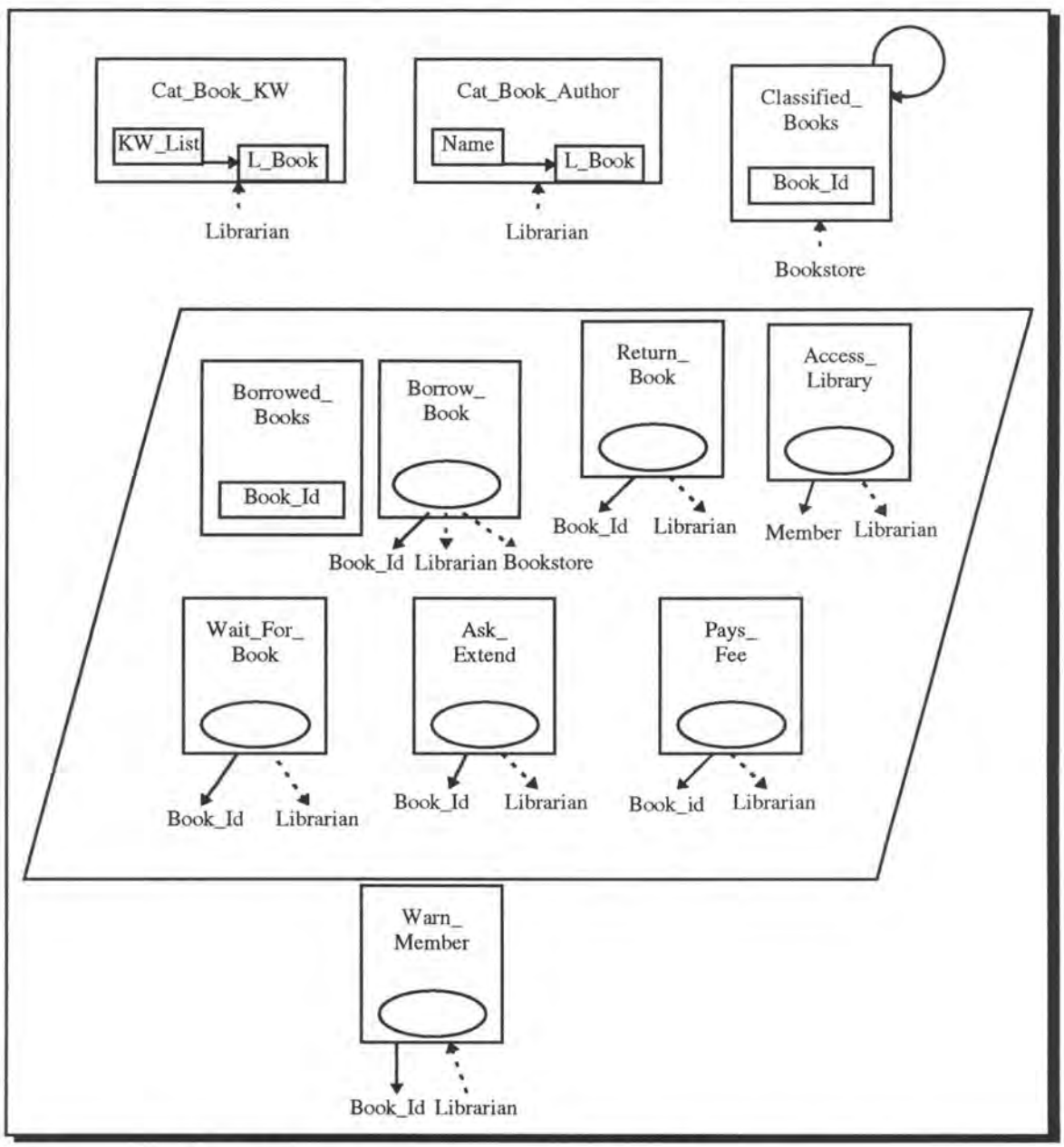


Figure 4.8 : Graphical declaration of the Ordinary Member agent

All the actions and the state components of the Ordinary member have already been described.

ORDINARY MEMBER

BASIC CONSTRAINTS

- **Initial valuation**

Card(Borrowed_Books) = { }

LOCAL CONSTRAINTS

- **State behaviour**

Card(borrowed_Books)≤ 2

- **Effects of action**

Return_Book(b) : Borrowed_Books = remove(Borrowed_Books, b)
Borrow_Book(b) : Borrowed_Books = add(Borrowed_Books, b)

- **Causality**

Borrow_Book(b) → Pays_Fee(b)

- **Capability**

XO(Return_Book(b)/in (Borrowed_Books, b))
XO(Borrow_Book(b)/in (Bookstore.Classified_Books, b))
XO(Ask_Extend(b)/in (Borrowed_Books, b)
F(Wait_For_Book(b)/in (Borrowed_Books, b)
F(Borrow_Book(b)/card(Borrowed_Books) = 2)

COOPERATION CONSTRAINTS

- **State perception**

ℵK(Librarian.Cat_Book_KW / TRUE)
ℵK(Librarian.Cat_Book_Author / TRUE)
ℵK(Bookstore.Classified_Books / TRUE)

- **Action information**

$\mathcal{K}$K(Return_Book(b).Librarian / TRUE)
$\mathcal{K}$K(Access_Library.Librarian / TRUE)
$\mathcal{K}$K(Pays_Fee(b).Librarian / TRUE)

Declaration of the Extra Member agent

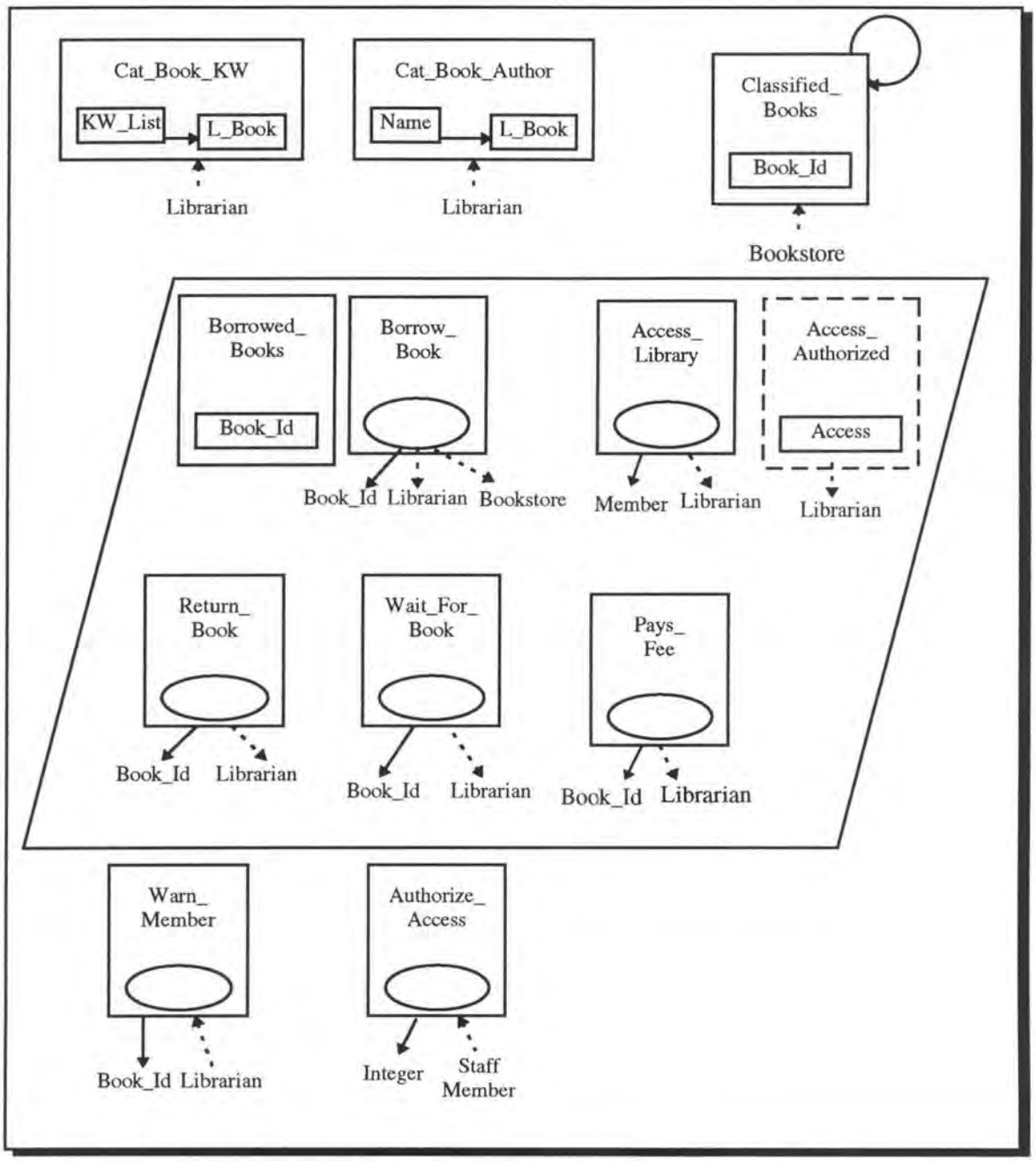


Figure 4.9 : Graphical declaration of the Extra Member agent

As for the Ordinary member, all the elements of the Extra member structure have been described in the previous declarations.

EXTRA MEMBER

BASIC CONSTRAINTS

- **Initial valuation**

Card(Borrowed_Books) = { }
Access_Authorized = FALSE

LOCAL CONSTRAINTS

- **State behaviour**

Card(borrowed_Books)≤ 2

- **Effects of action**

Return_Book(b) : Borrowed_Books = remove(Borrowed_Books, b)
Borrow_Book(b) : Borrowed_Books = add(Borrowed_Books, b)
m.Authorize_Access : Access_Authorized = TRUE

- **Causality**

Borrow_Book(b) → Pays_Fee(b)

- **Capability**

XO(Return_Book(b)/in (Borrowed_Books, b))
XO(Borrow_Book(b)/in (Bookstore.Classified_Books, b))
XO(Ask_Extend(b)/in (Borrowed_Books, b)
F(Wait_For_Book(b)/in (Borrowed_Books, b)
F(Access_Library/Access_Authorized = FALSE)
F(Borrow_Book(b)/card(Borrowed_Books) = 2)

COOPERATION CONSTRAINTS

- **State perception**

$\mathcal{K}$K(Librarian.Cat_Book_KW / TRUE)
$\mathcal{K}$K(Librarian.Cat_Book_Author / TRUE)
$\mathcal{K}$K(Bookstore.Classified_Books / TRUE)

- **Action information**

$\mathcal{K}$K(Return_Book(b) . Librarian / TRUE)
$\mathcal{K}$K(Access_Library . Librarian / TRUE)
$\mathcal{K}$K(Pays_Fee(b) . Librarian / TRUE)

- **State Information**

$\mathcal{K}$K(Access_Authorized.librarian/TRUE)

- **Action Perception**

$\mathcal{K}$K(m.Authorize_Access(i)/TRUE)

Declaration of the Bookstore agent

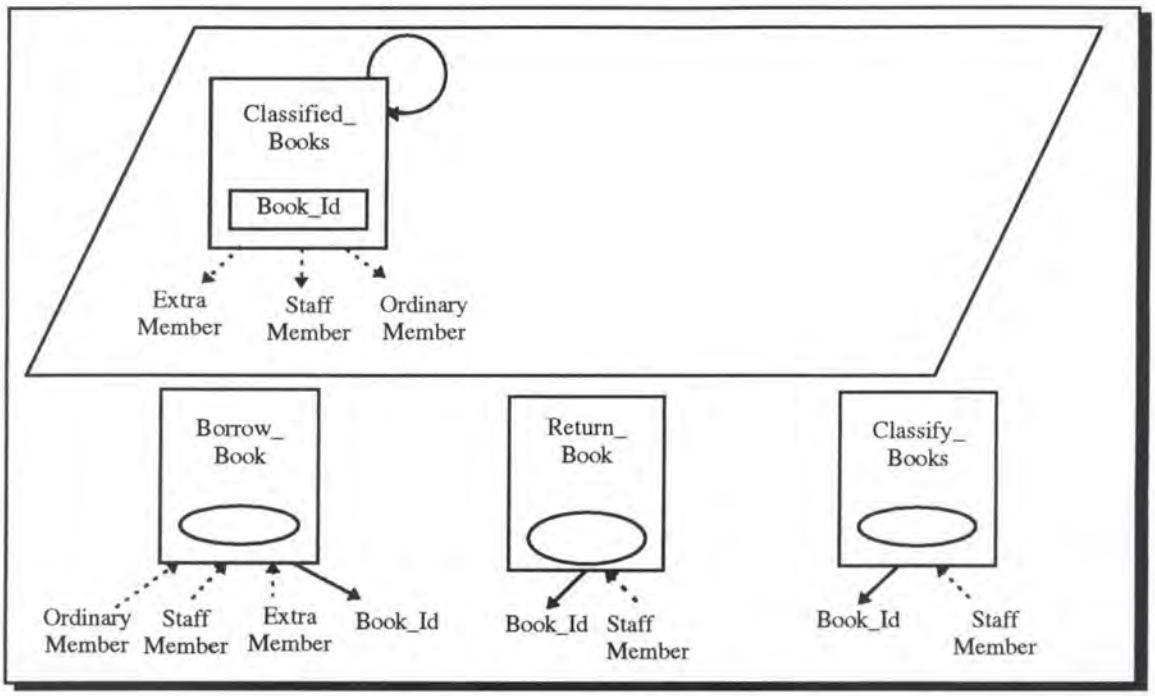


Figure 4.10 : Graphical declaration of the Bookstore agent

Constraints

Bookstore

LOCAL CONSTRAINTS

- **Effects of action**

m.Borrow_Book(b): Classified_Books = Remove(Classified_Books, b)

m.Return_Book(b): Classified_Books = Add(Classified_Books, b)

m.Classify_Books(b): Classified_Books = Add(Classified_Books, b)

COOPERATION CONSTRAINTS

- **Action Perception**

XK(m.Borrow_Book(b)/TRUE)

XK(m.Return_Book(b)/TRUE)

XK(m.Classify_Book(b)/TRUE)

- **State Information**

$\mathcal{K}$K(Classified_Books.m/TRUE)

4.3.2. i* specification

a) Agents identification

The agents identified for the application of the i* famework to the Library system before the introduction of any change in the organization, is virtualy completely the same. Nonetheless, only the terminal agents will be taken into account.
The Strategic Dependency models of the acquisition and the loan policies will thus show the dependencies between the Librarian, Staff Member, Ordinary Member, Extra Member and the Bookshop actors.

b) Acquisition policy

Strategic Dependency Model

We will now model the acquisition policy of the library by a strategic dependency model. We saw that this model aimed at supporting the dependencies between actors and that these dependencies can be of four different types. We'll try step by step, to find out who depends on whom and for what. For each dependency link on figure 4.11, we'll explain why we have choosen a specific type of dependency tather than another.

First of all, it is obvious that the main goal of the acquisition policy is just the ordering of a new book for the library. The Staff member actors are responsable for the good contents of the library and therefore choose which books to order, but they are not allow to issue the order by themselves; instead, they have to ask the Librarian to manage the book ordering. This kind of dependency should be model by a goal dependency (*Ordered (Book)*) between the staff member and the librarian.

The Librarian wouldn't be able to issue an order to a bookshop if he/she doesn't know which book to order. The librarian will thus depend on the staff member for the identification of the book. We're just talking about a simple information, and one could represent this by a resource dependency (*Book_Id*) between the Librarian and the staff member.

The identification of the book isn't enough to allow the librarian to issue the order. He/she will ask a staff member to pay the new book. This dependency is rather a task dependency (*Pays (Book)*) instead of a goal one because this dependency could be viewed as a constraint.

Now, the librarian is able to order the book to a bookshop. He/she will depend on the bookshop to have the ordered book, and this will be visualized by a goal dependency (*Received (book)*) between the librarian and the bookshop, because the librarian doesn't care how the bookshop will achieve this goal. The librarian is just interested by the result of the dependency.

Of course, the bookshop won't be able to send anything to the librarian if the latter doesn't send him an order. As for the dependency for the book identification between the librarian and the staff member, we will represent this by a resource dependency (*Order*).

The librarian depends on the staff member for the payment of the ordered book. However, as the librarian is the intermediate between the staff member and the bookshop, the latter will also depend on the librarian for the payment of the book. Therefore, we'll have the same task dependency link (*Pays (Book)*) between the librarian and the bookshop.

Once the librarian will have received the book, he/she will have to determine the reference of this book and for this reason, he/she'll ask the staff member to determine a category and a list of keywords to associate with the book. We have choosen to represent this by a task dependency (*Find-KW-Cat (Book_O)*) between the librarian and the staff member instead of a resource dependency because we want to express the fact that the staff member will have to carry out an activity.

Once the librarian will have register the identification and the reference of the new book, he/she will depend on the staff member to classify it on a library shelf. We have a goal dependency (*Classified (Books)*) between the librarian and the staff member because the librarian doesn't know how the staff member choose to classify the new books.

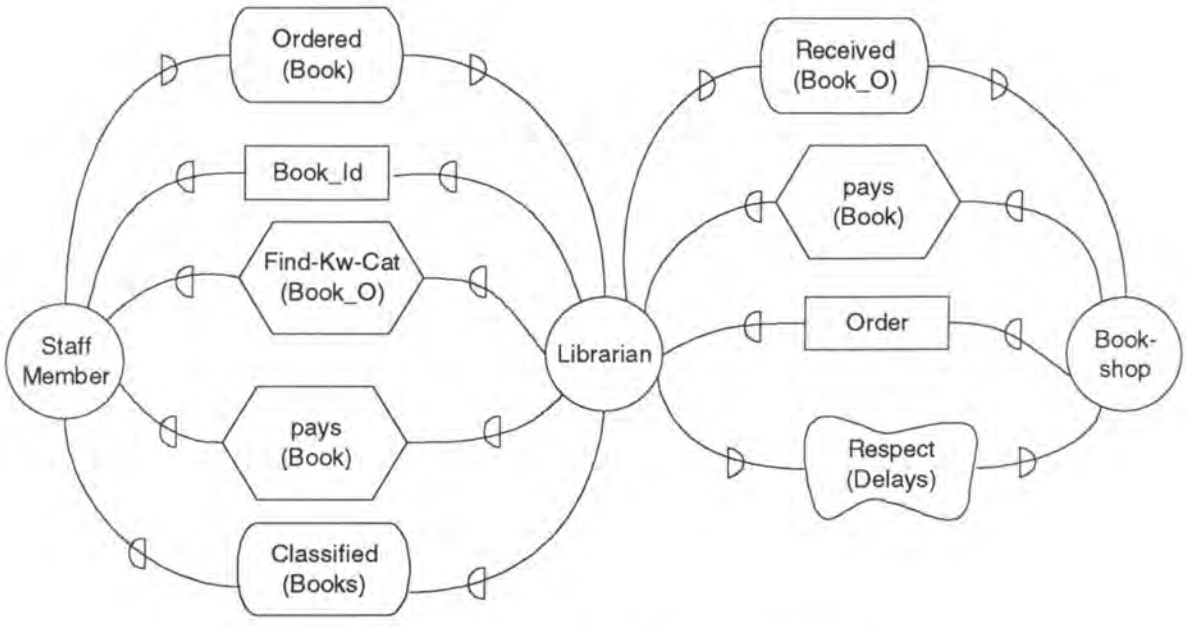


Figure 4.11: SD Model Of the acquisition policy

Strategic Rationale Model

Figure 4.12 showed us the different dependency links between the three actors implicated in the library acquisition policy. This SD model shows that the librarian is a central actor in the acquisition procedure. He/she plays a role of intermediate between the staff members and the bookshops. The staregic rationale model can help us to carry out a deeper analysis of the librarian role, and to find out some alternatives that could improve the actual way of working.

We know that the labrarian only issues a book order on a staff member request. This will be modelled in the Strategic rationale model by a goal dependency (*Ordered (Book)*) from the staff member actor into the librarian boundary.

The librarian has to order a book, this is represented by the (*order (book)*) task. This task consists of two components: the subtask of establishing the order (*Establish(order)*) and the subtask of receiving the new book (*Manage reception(book)*). We are only modelling the tasks that are considered important enough to be of strategic concern to the actor.

In order to establish a book order, we saw in the strategic dependency model (figure 4.11), that the librarian needed the book identification. Once again we will represent this by a resource dependency (*Book-Id*) going from the librarian boundary to the staff member actor. The task (*establish (order)*) will be also composed of two subtasks: (*Issue (order)*) and (*Register (order)*).

One way to have the order registered (which is the actual way of working), is to update the order list manually. We will represent this by the means end link (*Register Manually (order)*). But all along the book acquisition process, the librarian is influenced by two qualitative goals. On one hand, he/she wants to be *efficient*. He/she also wants to evolve in an adequate environment, with practical working methods, we will call this to be *comfortable*.

The fact of register an order (or anything) manually, certainly affects negatively the two qualitative goals. It represents a lost of time, a risk of lost, a possibility of data redundancy,...
The alternative could requires the use of an information system for the data registration.
This is depicetd by the means-ends link (*use IS*). In these circumstances, the librarian would depend on the IS for the data gathering or consultation.

When a librarian receives a new book from a bookshop, he/she has to register the book in the library books catalogues. The librarian will first ask the category and a list of keywords to the staff member (*Cat-Kwds*), and then updates the catalogues. One could make here the same reasoning as for the registration of an order. Either the librarian handles the old catalogues and makes the registration manually, either he/she uses an information system. In order to keep the SR model as clear as possible, we havn't drawn the links affecting the qualitative goals.

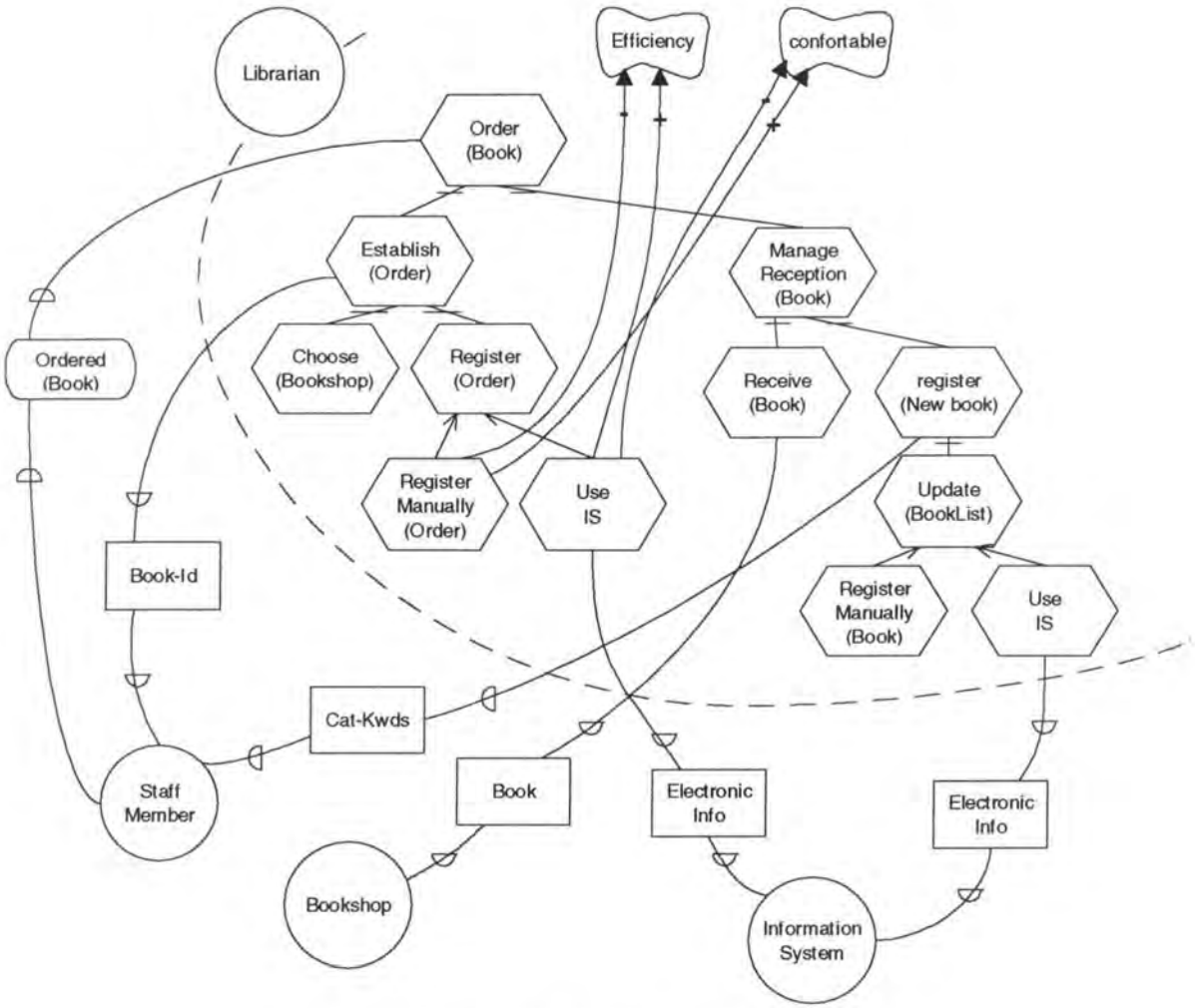


Figure 4.12: SR Model of the acquisition policy

c) Loan policy

Strategic Dependency Model

We will describe the loan policy the same way we did for the acquisition policy. The SD Model is depicted on figure 4.13.

All the users of the library need to be registered before his/her first access to the library. The registration is made by the librarian. So, if a user wants to borrow a book, he/she will first depend on the librarian to be registered. We will represent this by a goal dependency (*Register (member list)*) between the staff, ordinary and extra users and the librarian. We'll use a goal link because the users are just intrested in the result - beeing registered and thus, authorized to acces the library - and not by the procedure the librarian will have to follow in order to register them.

If he/she wants to register a new user, the librarian will need some information which we will represent by a resource dependency (*Registration info*) between the librarian and the users (ordinary, staff and extra members).

We will now tackle the heart of the loan policy, that is to say, the loan of a library book. The extra and ordinary users depend on the librarian for borrowing a book. The librarian is the one who manages all the loan procedure. The goal of these users is very simple, they just want to borrow a book and doesn't care about the updating of the loans list etc. It's the librarian's business, that's why we will represent the loan of a book by a goal dependency (*Loaned(book)*) between the extra and ordinary member actors and the librarian. The staff member doesn't have that kind of dependency with the librarian because he/she can borrow a book without asking the librarian.

But an extra or ordinary user will be able to borrow a book only if he/she pays a fee to the librarian. We thus have a resource dependency (*Fee*) between the librarian and the extra and ordinary member actors. Once again, we won't see that kind of dependency between the librarian and the staff member because the loan is free for the latter.

When a user (extra or ordinary member) borrows a book, he/she is supposed to keep it for a given period of time and not more. The librarian has to trust the user when he/she loans the book and will thus depend on him/her for the respect of the loan time. We choosed to represent this dependency by a softgoal link (*Respect (Loan time)*) between the librarian and the extra and ordinary members.

In the library description, we also saw that the staff member could evict the users who didn't behave correctly. In order to avoid that kind of situation, the staff member will depend on the extra and ordinary members for having an « adequate » behaviour. This is obviously a softgoal dependency (*Behave correctly*) because the meaning of « adequate » is not clear-cut defined.

If an extra member wants to borrow a book, a staff member will first have to authorize this extra member to enter the library. The extra member depends on the staff member to be allowed to access the library. This will be represented by a goal dependency (*Authorized(access)*) because of the extra member ignorance of the procedure to follow in order to receive this authorization.

An ordinary member is allowed to borrow a book for a period of fifteen days. An extra member doesn't have the same loan period. Actually, his/her loan period is given by a staff

member. The extra member has thus a resource dependency (*Loan time*) on the staff member actor for the loan length.

Finally, as for the acquisition policy, the librarian depends on the staff member for the ranking of the returned books in the library shelves. This is represented by the goal dependency (*Classified (books)*).

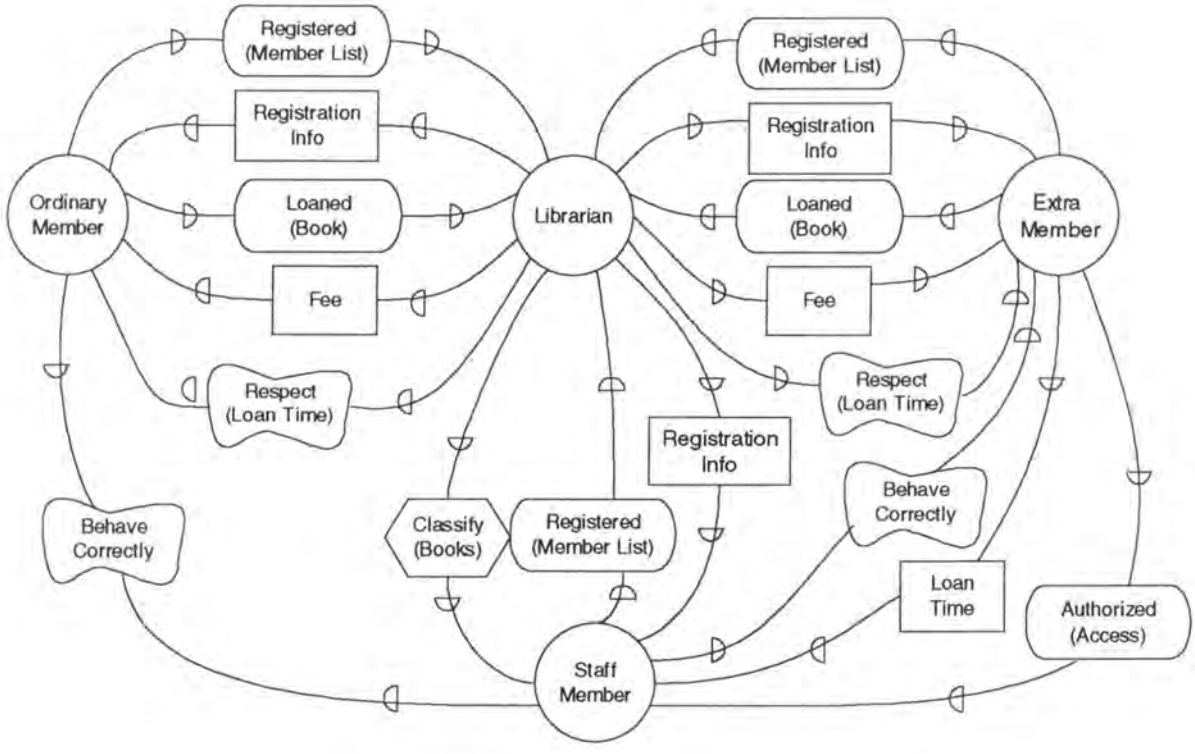


Figure 4.13: SD Model of the loan policy

Strategic Rationale Model

We saw that one of the main goals of the librarian was to manage book loans. In order to acheive this goal, the librarian has to perform three main tasks which are: the rent of books, the management of the returned books and he/she also has to manage the delays in returning books.

At this state of the reasoning, one should wonder what seems important to the librarian. What are his/her internal goals? What would he/she want to achieve? We wouldn't be wrong if we said that the librarian wants the library business prosperous.

The question one should then ask to ourselves is: what would make a library system prosperous? The answer would probably be the same for most of the businesses: an efficient management in one hand and customers happy in the other hand. These two parameters would affect positively the success of the business.

If we go on with this kind of reasoning, one could ask what makes a library user happy? Probably some flexibility for the loan period with for exemple the possibility to extend the loan period if nobody else's waiting for the book, reasonable loan periods. A user doesn't want either to wait to long for a book if it's already borrowed.

Of course, these two qualitative goals are opposed to each other and one will have to make the balance between them.

In the loan policy described earlier, we saw that the librarian didn't discover delays systematically but rather at random. He/she discovered that a borrower was late in returning a book when another borrower asked for the same book. This way of working could please some users, taking this opportunity to keep a book for a longer time, but in the other hand, the chances of beeing on a waiting list increase.

An alternative to this way of working could be the management of the loans delays by an information system.

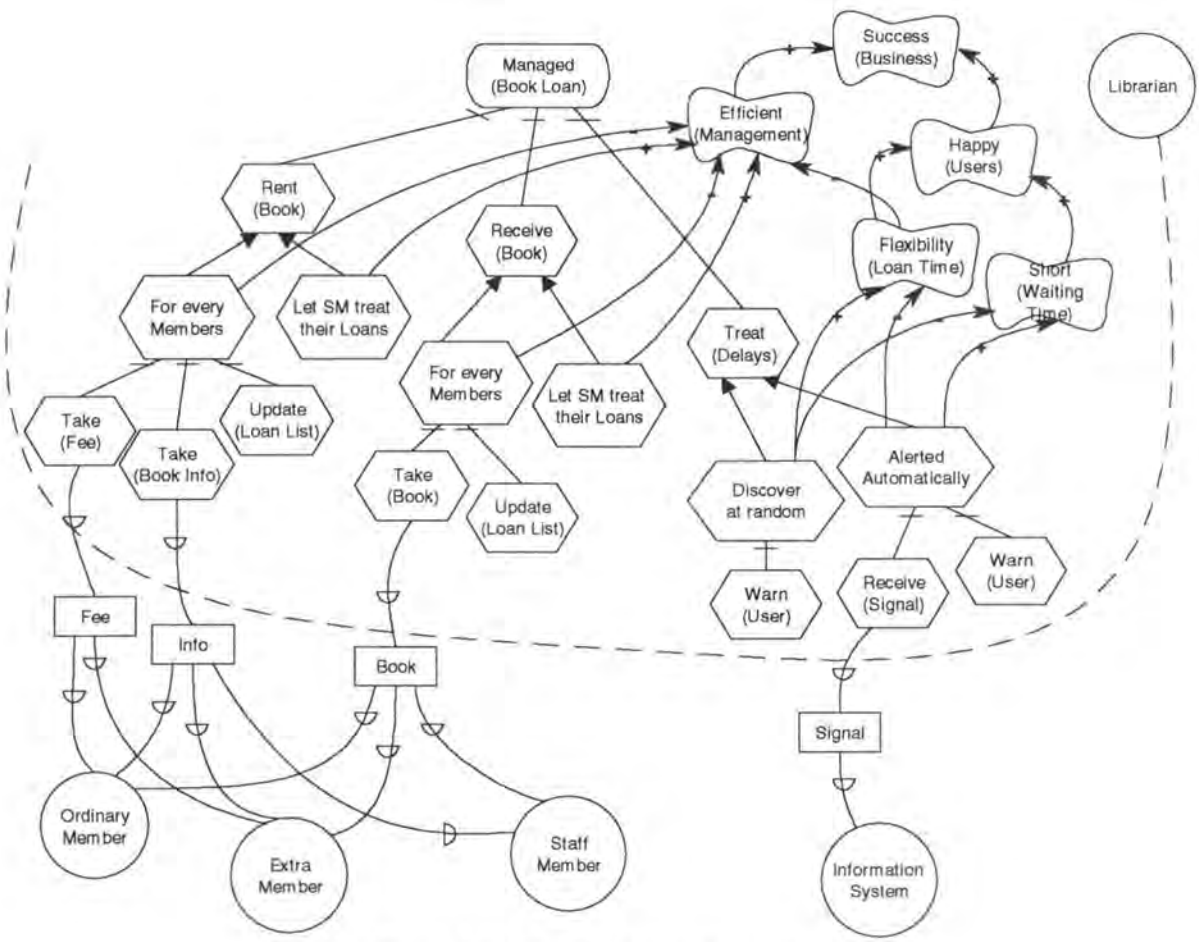


Figure 4.14: SR Model of the loan policy

Random wouldn't have its place anymore and the librarian would be warned systematically for the negligent borrowers.

The extra, ordinary and staff members depend on the librarian for the loan of books. When the librarian rents a book, he/she has to ask a fee (if the borrower is an ordinary or an extra member), to take some information (as the book identification) in order to update the loan list. The librarian is the only one who is authorized to update the loan list. Maybe this way of working isn't the more adequate or the more effective?
One could imagine the staff members more independant and one way to achieve this, could be to let them manage their own loans. They would have access to the information system and could update the loan database by themselves, this way, the staff members would have a little more responsibilities and the librarian would gain some time for his/her other tasks.

The Strategic Rationale Model helped us to find some alternatives in order to make the actual loan process more efficient. it showed that this could be achieved by the introduction of an information system and by the shift of responsibilities.

4.4. Introduction of the Information System

4.4.1. IS description

The new system to design should integrate the following facilities derived from the analysis made in section 4.3:

A book database: the system must record any relevant information about the books contained in the library: it must be able to answer enquiries about books appropriatly: given some criterion (author(s), keyword(s), title, year of publication,...), find if there are books matching the criterion in the library and, if yes, find their locations and status (borrowable, borrowed, lost).

An order database: the system must keep track of all the books orders in progress; it must be able to answer questions about the orders, such as: give the list of all the orders in progress, who has ordered a given book, check wether a book has been ordered, retreive pending orders issued for one month or more.

A member database: the system must record information about the library members such as the identity, the member category, where the member can be contacted.

A loan database: the system must record all the book loans (which book, which borrower, the date of the loan) and must be able to answer enquiries such as: who has a given book, give all the books borrowed by a member, give the list of all members being late in returning books, who is waiting for a borrowed book.

The book database can be consulted by any member. Staff members can consult any database. An ordinary or extra member may not receive information about other members. The librarians are the only people allowed to update the databases. There is an exception however: any staff member can update the loan database for his/her own loans.

4.4.2. i* Description

a) New agents identification

The actors of the new library system are identical to the terminal agents identified for the application of the Albert language (except for the shelf agent). The usual actors will be:
the Staff Members, the Ordinary Members, the Extra Members and the Librarian;
We introduce a new actor: the Information System (IS).

b) Acquisition policy

Strategic dependency model

The introduction of an information system means the introduction of a new agent, and who says new agent, says automatically new dependencies. These new links are underlined with a new strategic dependency model depicted on figure 4.15. Once again, we'll argue the choice of the new links between the actors.

For the acquisition policy, the main dependency links remain the same. Nonetheless, the librarian and the staff member actors will depend on the new actor, the information system, in order to consult the library databases. We'll represent this by a simple resource dependency link (*Information*) between the staff member and the IS and between the librarian and the IS.

In its turn, the IS will depend on the librarian for the updating of its information. We have a task dependency (*update (info)*) because the procedure is well established. There is only one way to update an information system.

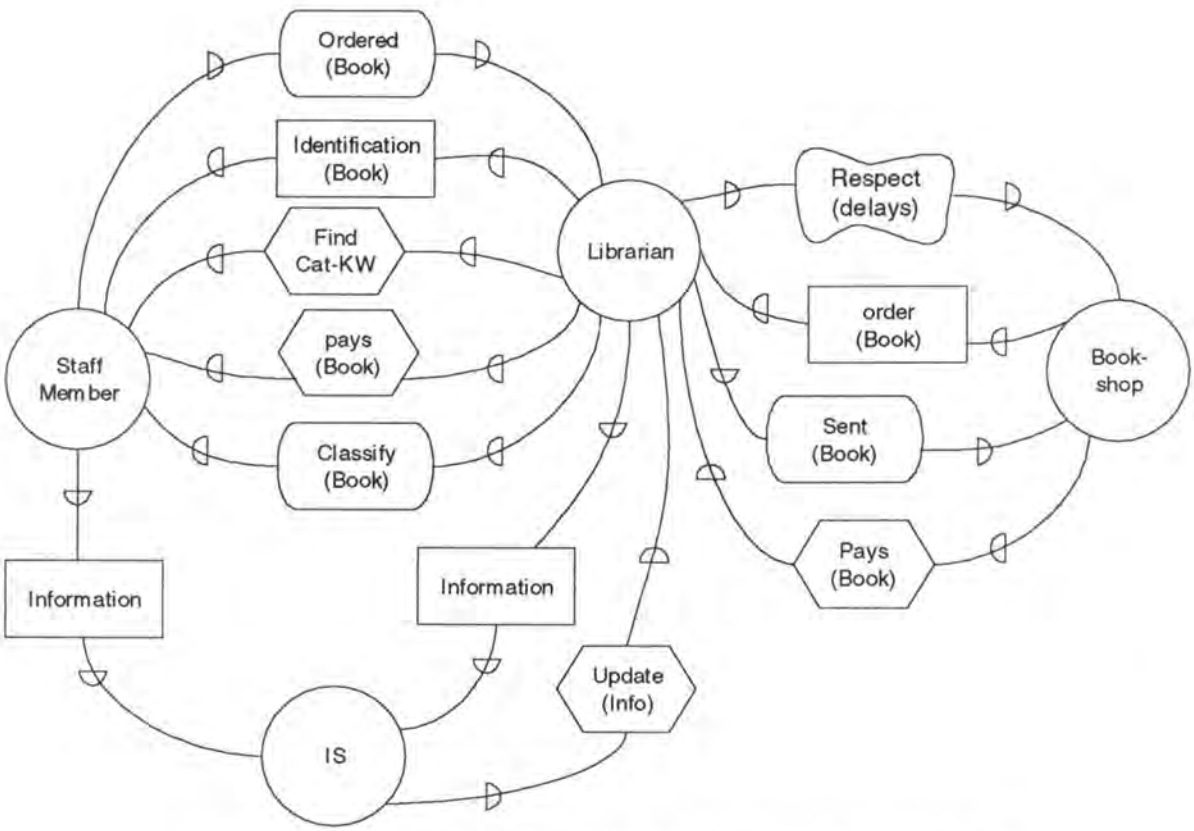


Figure 4.15: SD Model of the acquisition policy after introduction of the IS

c) Loan policy

We'll show via the SD model depicted on figure 4.16 that the introduction of the information system has modified a part of the dependencies between the actors of the loan process.

Nonetheless, some of the dependencies remain the same (*Registered(member-list)*, *Registration info*, *Borrowed (book)*, *Fee (book)*) and we won't explain them anymore. Furthermore, in order to lighten the model, we've only represented the goal (*register(member-list)*) and the resource (*Fee(book)*) as dependencies between the ordinary member and the librarian but it's obvious that the same dependency links exist between the extra member and the librarian. We've done the same for the goal (*registered(member-list)*) and the resource (*registration info*); these dependency links exist also between the extra and staff member actors and the librarian actor.

We saw previously that the librarian discovered the delays more or less at random. Now, this actor depends on the information system to have systematically the list of negligent borrowers. We will represent this by a resource dependency (*Delays list*) between the librarian and the IS.

The librarian is also able to consult the databases of the IS, and thus will depend on it for answering to his/her enquiries. We have represented this by a goal dependency (*answered(enquiries)*) between the librarian and the IS because the librarian doesn't tell the IS how to answer. He/she is only interested by the outcome.

For responding to the librarian questions, the IS needs some criterion specifying the request. This is visualized by a resource dependency (*Criterion*) link between the IS and the librarian.

The IS needs also to be updated for each loan registration and will depend for this on the librarian for the extra and ordinary members loans, and it will depend on the staff member actor for his/her own loans. The procedure for updating the loan database is well established that's why wel'll represent this dependency by a task link (*Update(loan-DB)*).

Finally, the librarian will ask the new information system to have good performance. It will be represented by a softgoal (*good performance*) because we don't have a sharply defined definition of what is a « good performance ».

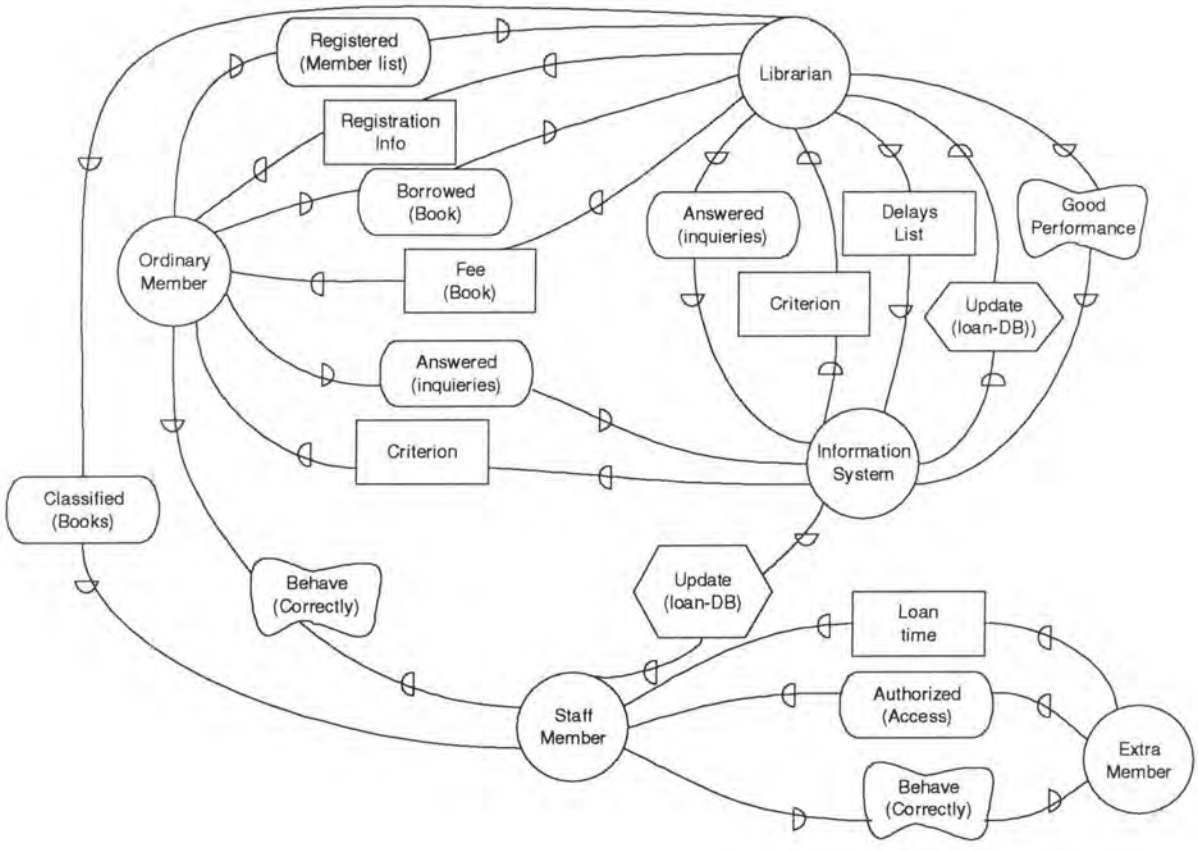


Figure 4.16: SD Model of the loan policy after introduction of the IS

4.4.3. Albert specification

a) New agents identification

The former agents remain the same as in the previous organization but the Library society has a new terminal agent: the Information System.
The new graphical declaration of the library system is shown figure 4.17

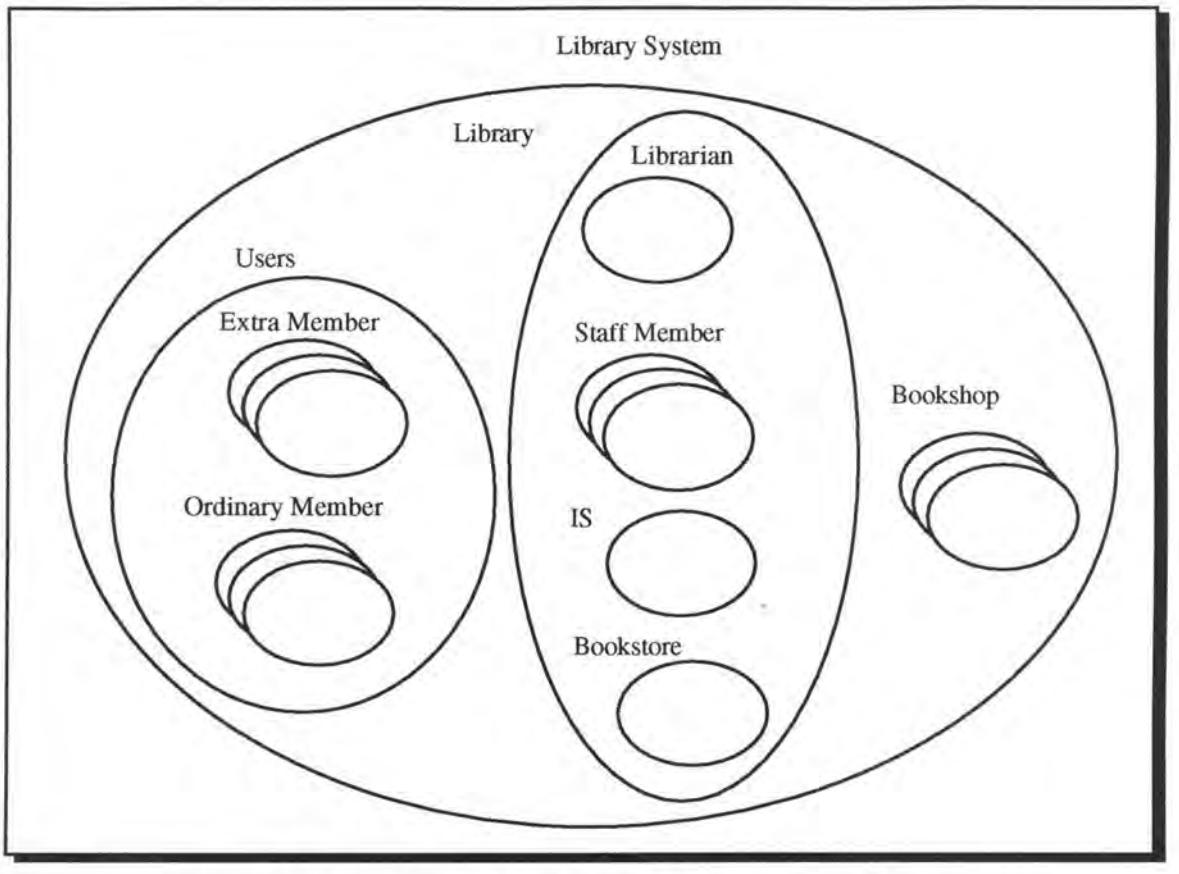


Figure 4.17 : Graphical declaration of the Library system society

b) Acquisition policy

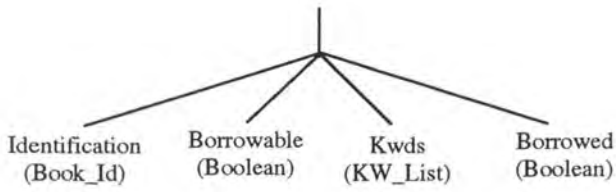
Declaration of the Staff Member agent

The new graphical declaration of the Staff Member after introduction of the Information System is depicted on figure 4.18. The main changes are the following:

1. The Staff Member agent has five external state components. Two of them have already been described (B_To_Classify and Classified_Books). This agent has now the perception of the Order database (Ord_DB), the Member database (Mem_DB) and the two book catalogues have been merge into one database (Book_DB).

Book_DB: this state component represents the set of books (Book) registered in the library.

- The Book type has been modifed. It is now defined by:



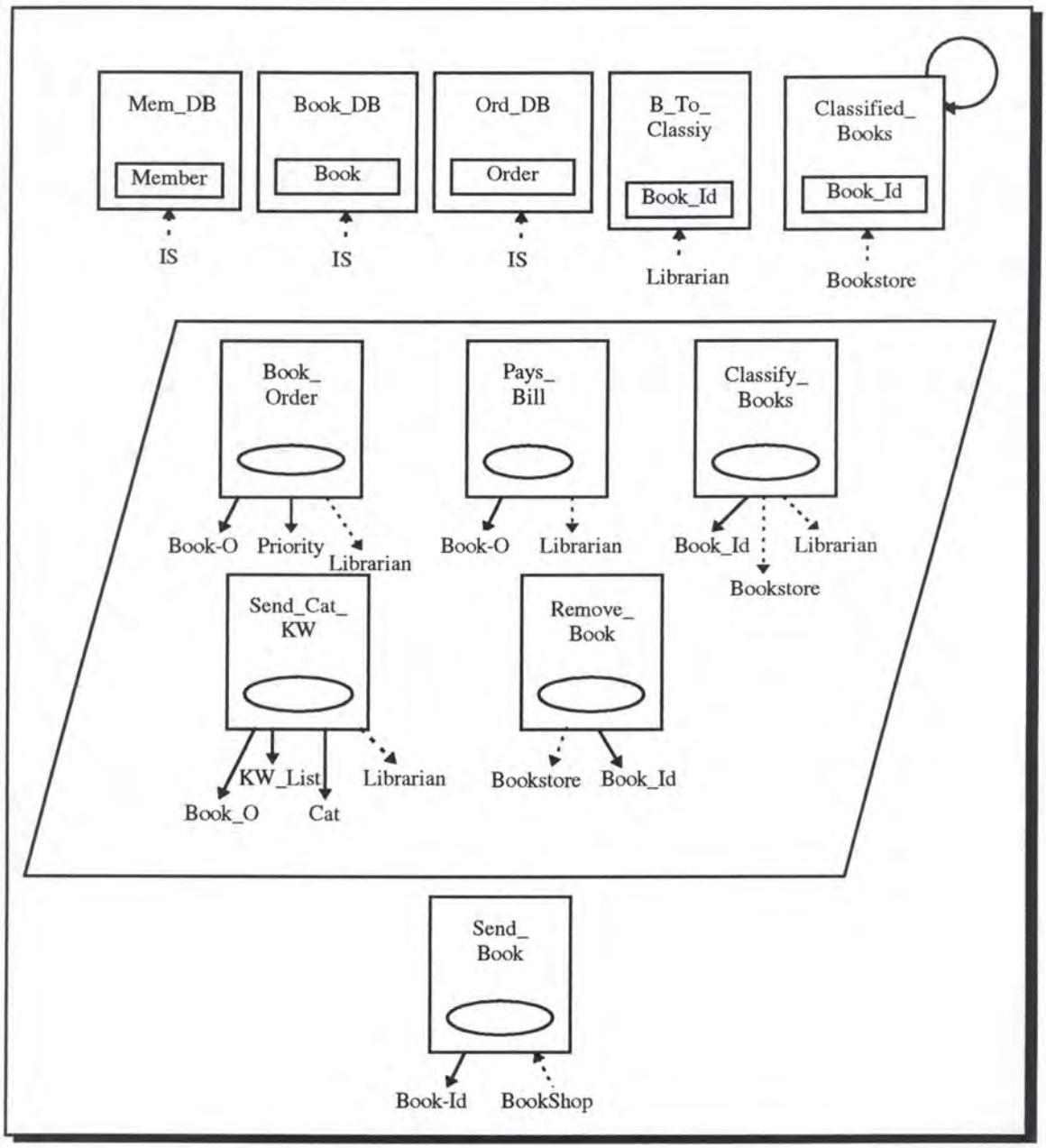
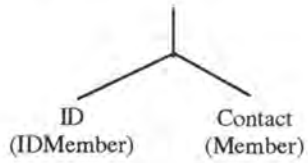


Figure 4.18 : Graphical declaration of the Staff Member agent

Mem_DB: This set represents all the members (Member) registered in the library (they aren't registered anymore in a table).

- The new type Member is the following:



STAFF MEMBER

LOCAL CONSTRAINTS

- **Causality**

Bookshop.Send_Book (b)→ Send_Cat_KW (b, kw, c)
** once a bookshop has sent a new book, a staff member sends to the librarian, the category and a list of keywords associated with that book*

- **Capability**

$\mathcal{XO}$ (Classify_Books(b) / ¬ in(B_To_Classify, b))
** a staff member can't classify a book which isn't in the "B_To_Classify" list*

$\mathcal{XO}$ (Remove_Book(b) / ¬ in(Classified_Books, b)
** a staff member can't remove a book which is not in the bookstore*

COOPERATION CONSTRAINTS

- **Action Perception**

$\mathcal{XK}$ (b.Send_Book (b) /TRUE)

- **State Perception**

$\mathcal{XK}$ (Bookstore.Classified_Books/TRUE)
$\mathcal{XK}$ (IS.Ord_DB/TRUE)
$\mathcal{XK}$ (IS.Book_DB/TRUE)
$\mathcal{XK}$ (IS.Mem_DB/TRUE)
$\mathcal{XK}$ (Librarian.B_To_Classify/TRUE)

** the staff members always perceive the catalogues, the bookstore and the B_To_Classify lists*

- **Action Information**

ℵK (Classify_Books(b).m /TRUE)
ℵK (Remove_Book(b).b/TRUE)

Declaration of the Librarian agent

We'll just describe the actions and state components that havn't been already described.

The Librarian agent has three new internal actions (Class_Order, Remove_Order and Add_Book_order):

Class_order: the Librarian asks the IS to register a new book order (Order) in the Ord_DB.

Remove_order: the Librarian agent would like to remove an order (Order) from the Ord_DB.

Add_Book_DB: the Librarian registers a new book (Book) in the IS Book database.

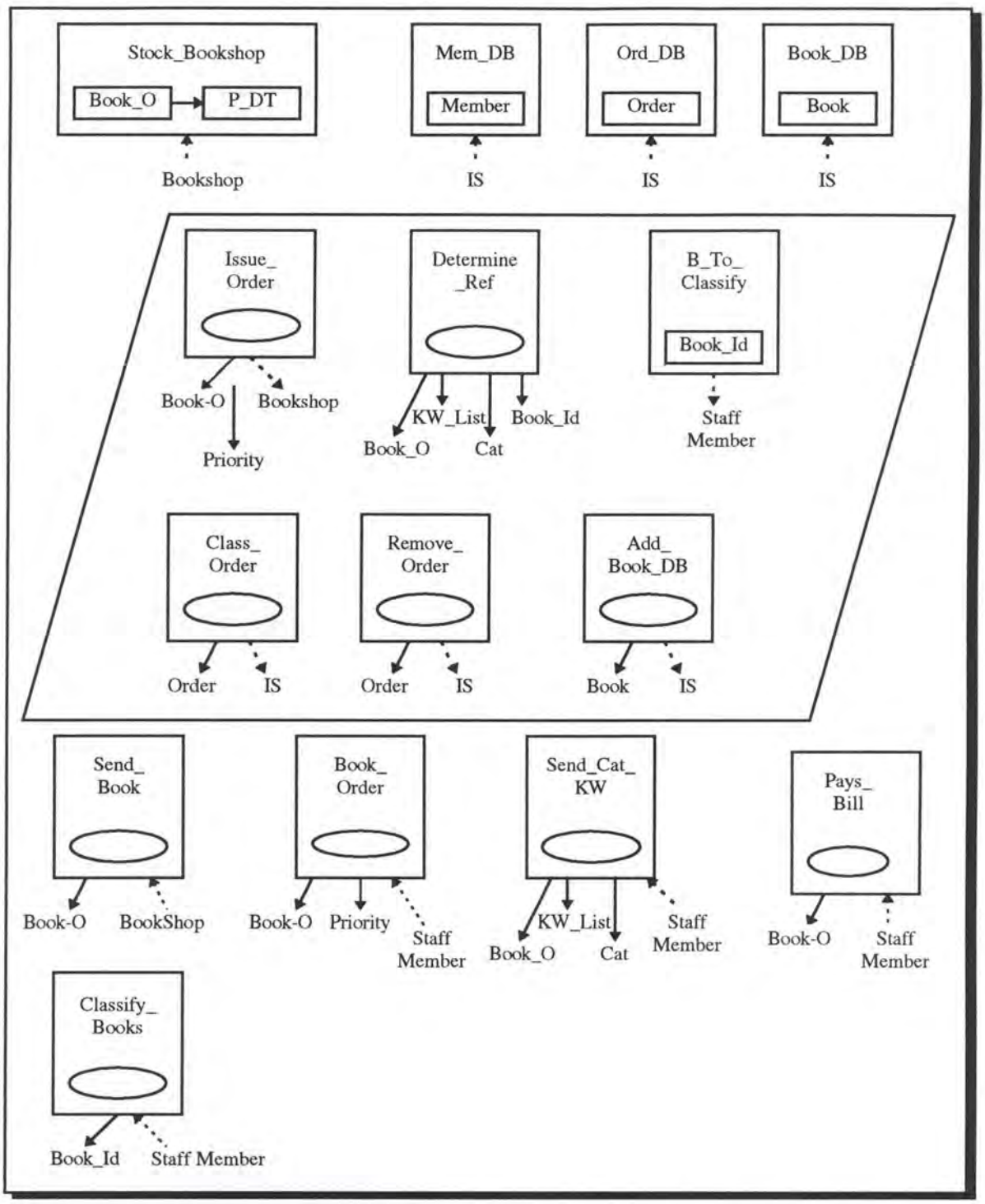


Figure 4.19: Graphical declaration of the Librarian agent

Constraints

LIBRARIAN

BASIC CONSTRAINTS

- **Initial Valuation**

B_To_Classify = { }
* *There are no books to classify in the library Bookstore*

LOCAL CONSTRAINTS

- **State Behaviour**

in (B_To_Classify, b) $\Rightarrow_{>1day}$ $\neg$ in (B_To_Classify, b)
* A *book must be classified in the library Bookstore within one day.*

- **Effect of Actions**

Determine_Ref(bo, kwl, c, b): B_To_Classify = add(B_To_Classify, b)
* *When the librarian has determine the reference of the new book, he/she puts it in the B_To_Classify list*

Staff Member.Classify_Books (b): B_To_Classify = remove (B_To_Classify ,b)
* *When a Staff Member classifies a book in the library, he/she removes it from the list "B_To_Classify".*

- **Causality**

Staff Member.Book_Order (b, pr) ; _ . Pays_Bill (b) → Issue_Order (b, pr).bs;
Class_Order(O)
with Book_O(O) = b
Member(O) = Staff Member
Date(O) = Todays_Date
Bookshop(O) = bs

* *When a Staff Member asks the librarian to order a book(b) at a given priority and if a Staff Member agrees to pay the bill for that book, then the librarian will issue an order to the appropriate bookshop and will register the order in the orders list.*

Staff Member.Send_cat_KW(b, kw, c) → Determine_Ref (b, c, book)
with Authors (book) = Authors (b)
Title (book) = Title (b)
Edition (book) = Edition (b)
Category (book) = c;
Add_Book_DB(b)
with Identification(b) = book
Kwds(b) = kw
Borrowed(b) = FALSE

** When a staff member sends the category and a list of keywords associated with a book, the librarian can determine the reference of that book and can register it in the IS Book database*

Bookshop.Send_Book (b) → Remove_Order (O)
with Book_O(O) = b

** When a bookshop has sent a book, the librarian removes one order he chooses from the orders list which asked for that book*

- **Capability**

F (Issue_Order(b, p).bs / (Isof p = "price" ∧
∃ bs' : Price(bs'.Stock_Bookshop[b])
< Price(bs.Stock_Bookshop[b]))
∨ (Isof p = "Del_Time" ∧
∃ bs' : Del_Time(bs'.Stock_Bookshop[b])
< Del_Time(bs.Stock_Bookshop[b]))

** the librarian can't send an order to a bookshop who doesn't respect the priority in time or in delivery time*

COOPERATION CONSTRAINTS

- **Action Perception**

I (Bookshop.Send_Book(b) / ¬ in (IS.Ord_DB, O): book_O (O) = b
∧ bookshop (O) = bookshop)

** the librarian ignores the book sent by a bookshop, which hasn't been ordered*

I (Staff member.Classify_Books (b) / ¬ in (B_To_Classify, b))

** the librarian ignores the action of a staff member consisting in removing a book that doesn't exist in the "B_To_Classify" list*

- **State Perception**

$\mathcal{K}$ (b.Stock_Bookshop[-]/TRUE)
$\mathcal{K}$ (IS.Mem_DB/TRUE)
$\mathcal{K}$ (IS.Ord_DB/TRUE)
$\mathcal{K}$ (IS.Book_DB/TRUE)
** the librarian can always perceive the catalogues of the bookshops*

- **State Information**

$\mathcal{K}$ (B_To_Classify.Staff Member/TRUE)
* the "B_To_Classify" list is always visible to the staff members.

- **Action Information**

$\mathcal{K}$ (Issue_Order(b, p).Bookshop/TRUE)
$\mathcal{K}$ (Class_Order(o).IS/TRUE)
$\mathcal{K}$ (Remove_Order(o).IS/TRUE)
$\mathcal{K}$ (Add_Book_DB(b).IS/TRUE)

Declaration of the IS agent

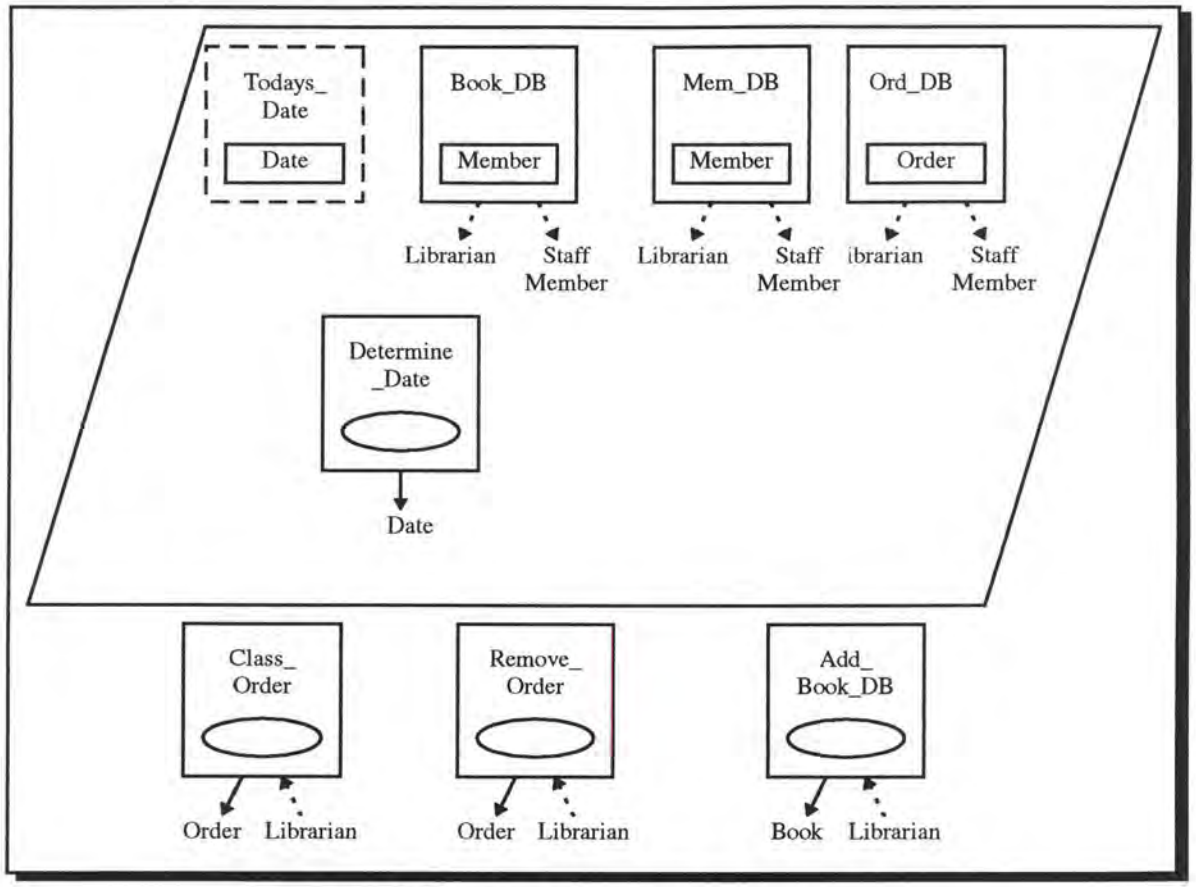


Figure 4.20: Graphical declaration of the IS agent

Constraints

IS

BASIC CONSTRAINTS

- **Initial Valuation**

Ord_DB = { }

* *There are no orders in progress in the orders database*

LOCAL CONSTRAINTS

- **Effect of Actions**

Librarian.Class_Order(O): Ord_DB = Add(Ord_DB, O)

Librarian.Remove(O): Ord_DB = Remove(Ord_DB, O)
Add_Book_DB(b): Book_DB = Add(Book_DB, b)
Determine_Date(d): Todays_Date = d

COOPERATION CONSTRAINTS

- **Action Perception**

I (Librarian.Remove_Order(O) / ¬ in(Ord_DB, O))
ℵK (Librarian.Class_Order(O) / TRUE)
ℵK (Librarian.Add_Book_DB(b) / TRUE)

- **State Information**

ℵK (Book_DB.m / TRUE)
ℵK (Mem_DB.m / TRUE)
ℵK (Ord_BD.m / TRUE)

Declaration of the Bookshop agent

The declaration and the constraints of this agent are the same as figure 4.4.

Declaration of the bookstore agent

The declaration and the constraints of this agent are the same as figure 4.5

c) Loan policy

The loan policy after the introduction of the information system has been completely specified in the annex.

4.5. Correlation between Albert and i*

We have just applied to the concrete example of a library case study, the two frameworks Albert and i*, developed in chapters two and three. Each of these methods has its own objectives and characteristics.
In this chapter, following the reverse engineering method, we have first applied the Albert language to the acquisition and loan policies of the library, allowing us to understand in a formal way, the functionalities (the " what") of the studied work processes. We've then applied the i* framework to the existing system in order to discover and to identify the dependencies existing between the main actors. The Strategic Rationale model of the framework has allowed us to emphasize some of the work procedure weak points such as the delays detection. This model also helped us to find out new alternatives in order to improve the present situation.
The introduction of an information system has overturned the dependencies and relationships within the library system, that's why we used again the Strategic Dependency model in order to catch on this brand new organization. Last but not least, we have modelized the new system the same way we did at the begining, i.e. by the application of the Albert language.

The main goal of this section is to discover if it could be possible to establish a closer link between the two frameworks. Could it be possible to settle a systematic binding between Albert and i*, allowing us to infer automatically the Strategic Dependency model from the formal Albert specification and conversely, to deduce systematically the Albert specification from the new SD model of the rehandled organization?
We'll first analyze the similarities and differences between Albert and i*, and thanks to these observations, we'll try to answer the question of knowing if a systematical binding could really be established.

4.5.1 Common points between i* and Albert

We've decided to underline thwo major similarities between the two methods:

- **The composite system aspect**

The two frameworks take the composite system notion into account. They are not restricted to an information system or whatever, to be developed but consider the whole context in which a new technological structure has to be embedded.

Albert will then identify the different **agents** involved in a system. Furthermore, it proposes the regrouping into agent societies in order to handle the real problems complexity. These agents

represent several elements of the system such as human beings, hardware, software pieces, etc. They can be external or internal to the studied system.
In the library example, the Albert specification has taken into account the librarian how was part and parcel of the library but also the bookshops which only had an indirect link with the library management.

The i* framework adopts a similar view. The Strategic Dependency model decomposes the considered system in several **actors**. If the society or population of agents notions are not depicted just as it is in Albert, the Agent-Role-Position model allows to refine the actors decomposition proposed by the SD model.

- **Communication between "actors" or "agents"**

The second similarity that we would like to approach in this section, is the communication between the system components. We explained the Albert and i* decompositions in agents or actors, but the existence of such elements without any kind of communication between them, wouldn't mean anything. The two methods express this communication in different ways:

Albert uses the mechanism of importation and exportation links between the components (state components or actions) of the agents structures, along with the cooperation constraints associated with each of them. It is therefore possible to answer some questions such as: which agent modifies another agent's behaviour and how, which agent is able to perceive the internal state components of another agent and when, etc.

I* will rather talk about actors dependencies instead of communication between them. The Strategic Dependency model depicts a work process in terms of dependency links and even refines this notion by identifying four types of dependencies (goal ,resource, task and softgoal). Looking at a SD model, one can answer some questions such as: who depends on whom and for what.

4.5.2. Main difference between Albert and i*

In the previous section, we saw that it exists a clear separation between the two frameworks, and each of them thaugth us something different about the library system.
From the analysis requirements specified by Albert, we learned for example, that a returned book had to be classified in the library shelves within one day or that an ordinary member could borrow a book for 15 days. We also learned what the librarian had to do in order to rent a book, what informations he/she needed.

The Albert specification gave us a clear view of **what** the system was doing along with temporal constraints, but it didn't tell us more. With that kind of specification, it's not always easy to understand systematicaly what could be wrong in the work process.
Introducing an information system at that stage of the analysis would lead to the mistake raized by M.Hammer: "Don't automate, obliterate". That's why, before talking about the introduction of any technologies, we tryed with i* to understand **why** the present process needed to be improve.
So, it is obvious that the two frameworks are located at two different levels of comprehension. Used together, these models lead to a complete undertsanding of the business process, that's why it would be very useful to find or to create a systematic binding between them. However, it seems yet very difficult establishing such a link.

4.5.3. Binding between Albert and i*

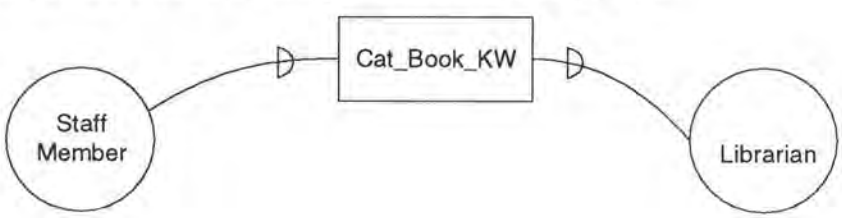
We'll analyse the binding idea at two levels: between all the agent declarations in Albert and the SD model of i* , and between one particular agent declaration in Albert and the SR model for that actor in i*.

From Albert specifications to the SD model (and reciprocally)

An intuitive way to induce automatically an SD model from Albert specifications for a same work process, could be the following:

- each agent represented in Albert would be represented by an actor in i*;
- each importation link for an action or a state component, from an agent a1 to an agent a2 could be represented by a dependency link with the actor a1 as the depender, the actor a2 as the dependee and the action or the state component as the dependum;
- each exportation link for an action or a state component from an agent a1 to an agent a2 could be represented by a dependency link with the actor a1 as the dependee, the actor a2 as the depender and the action or the state component as the dependum.

For example, from figure 4.2. presenting the librarian declaration for the book acquisition policy, one could find for the state component « Cat_Book_KW »:



Unfortunately, one will find out that different problems will occur nearly immediatly

1. The Albert language represents all the actions the same way, while the SD model proposes four types of dependum. The problem will then happen in the sense Albert to i*: how choosing the right representation for a particulal action?

2. It seems obvious that the state components of the Albert language will be represented by resources in i*. However, Albert has chosen to differentiate its state components representations according to their types (table, sequences, individual components,...), while i* doesn't make any difference between the resource dependencies. This time, the problem will occur in the sense i* to Albert: how choosing the right representation for a particular resource?

3. i* proposes softgoal dependencies that is to say dependum which are not clearly defined. This notion doesn't exist in Albert. On the contrary, the studied examples have to be complete and precise enough to include temporal or performance constraints. How deducing temporal or peformance constraints from an SD model? How infer softgoals from an Albert specification?

4. In Albert, some of the actions or state components internal to an agent declaration, have no importation or exportation links. It means that they won't be represented in the SD model and it's not a big deal in the sense Albert to i*, but how are we going to find them from an SD model?

5. This problem concerns the actions arguments in Albert. Some of them are not enough strategic to be represented in an SD model. (and which representation should we adopt?)

One could find many other problems compromising the binding aspiration between the two frameworks. In order to summarize the situation, one could say these problems are directly related to the fact that the i* framework is too subjective and not enough formal in at least two domains:

- first of all, the choice for example between a goal or a task for a same dependum is not always ovbvious and in most cases, this choice remains very subjective and has to be supported by convincing arguments.
- the choice of the dependencies that will be modeled, are also very subjective because only the dependencies that seem to be strategic enough for the analyst will be represented.

From one agent's declaration to the corresponding SR model (and reciprocally)

We've just seen that the binding between the Albert declaration of the agents and the SD model of the whole work process was very difficult to achieve. Most of the problems raised earlier can once again, be applied to the link between one agent declaration in Albert and the corresponding SR model for that particular agent.

Indeed, the SR model destined to catch the rationales inherent to a particular way of doing things, will only handel the rationales considered to be important enough to be modeled. It also represents qualitative goals which have no equivalent in the Albert language.

All these observations lead to the conclusion that at this stage of the i* life, it seems impossible to envisage a systematic binding with Albert. The i* framework is still too informal and conceded a too important place to the subjectivity of the analyst.

Chapter 5: Conclusion

For many companies, Business Reengineering is the only way to face the new client'demands and the concurrence context. In order to succedd in the introduction of an information system within a work process, it is really important to define and to understand adequately the work organization in which such technological support will be embedded.

In this thesis, we have presented two frameworks situated at two different levels in the requirements engineering activity. We have seen that the Albert language aimed at supporting the description of what a system is to do while the i* framework (via the SD and SR models), helpes us to catch the strategic dependencies and rationales inherent to the process actors.

Used together, these two approaches give a full comprehension of a business process but the idea of an automatical binding between them still seems to remain a sweet dream.
Indeed, the difference of levels and the lack of formalism of i* along with a large subjective counterpart still constitute an important obstacle.

Bibliography

[Agostini93] A. Agostini, G. De Michellis, M. A. Grasso, S. Patriarca, Reengineering a Business Process With an Innovative Workflow Management System: a Case Study, *Coocs'93*.

[Blomberg86] J. L. Blomberg, The Variable Impact of Computer Technologies on the Organization of Work Activities, *Proceedings of the conference on Computer-Supported Cooperative Work*, pp. 35-42, 1986, also in *Computer-Supported Cooperative Work - A Book of Readings*, I. Greif, ed., pp. 771-781.

[Briand94] L. Briand, W. L. Melo, C. Seaman, V. Basili, Characterizing and Assessing a Large-Scale Software Maintenance Organization-Experience Report-, *University of Maryland.*

[Bubenko80] J. Bubenko, Information modelling in the context of system development, *Proceedings IFIP World Congress*, 1980.

[Bubenko93] J. A. Bubenko, Extending the Svope of Information Modeling, *Proc. 4th Int. Workshop on the deductive Approach to information Systems and Databases*, Lloret-Costa Brava, Catalonia, Sept. 20-22, 1993, pp. 73-98.

[Chung94a] K. L. Chung, B. A. Nixon, E. Yu, Using Quality Requirements to Drive Software Development, ICSE-16 *Workshop on Research Issues in the Intersection Between Software Engineering and Artificial Intelligence*, International Conference on Software Engineering, Sorrento, Italy, May 16-20, 1994.

[Chung94b] K. L. Chung, B. A. Nixon, E. Yu, Using Quality Requirements to Systematically Develop Quality Software, *Fourth International Conference on Software quality*, October 3-5, 1994.

[Clement87] A. Clement, C. C. Gotlieb, Evolution of an Organization Interface: The New Business Department at a Large Insurance Firm, *Trans. Office Information Systems*, 1987.

[Curtis92] W. Curtis, M. I. Kellner and J. Over, Process Modelling, *Comm. ACM*, 35 (9), 1992, pp. 75-90.

[Dar93] R. Darimont, The Development Context in the ICARUS Process Model: Application to the Elicitation of KAOS Goals, *University of Louvain-La-Neuve (Belgium)*, Dec 1993.

[DDDP94a] E. Dubois, P. Du Bois, F. Dubru, M. Petit, The Albert Course Vol1: the Language, *University of Namur (Belgium)*, June 1994.

[DDDP94b] E. Dubois, P. Du Bois, F. Dubru, M. Petit, Agent-Oriented Requirements Engineering: A Case Study using the Albert Language, *University of Namur (Belgium)*, Septembre 94.

[DDP93] E. Dubois, P. Du Bois, M. Petit, Elicitating and formalising requirements for CIM information systems, *University of Namur (Belgium)*, june 93.

[DYDM95] P. Du Bois, E. Yu, E. Dubois, J. Mylopoulos, From Organization Models to System Requirements A « Cooperating Agents » Approach, *submitted to the 3rd International conference on cooperative Information systems CoopIS-95*, Vienna (Austria), May 9-12, 1995.

[Dub86] E. Dubois, J. Hagelstein, E. Lahou, A. Rifaut, F. Williams, Proceedings ESPRIT'85 conference, North Holland, 1986.

[Finkelstein] A. Finkelstein, A Course onRequirements Engineering.

[Greenspan94] S. J. Greenspan, J. Mylopoulos, A. Borgida, On Formal Requirements Modeling Languages: RML Revisited, (invited plenary talk*), Proc. 16th Int. Conf. Software Engineering*, May 16-21 1994, Sorrento, Italy, pp. 135-147.

[Hammer90] M. Hammer, Reengineering Work: Don't Automate, Obliterate, *Harvard Business Review*, July-August 1990, pp. 104-112.

[Jacobson] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard, Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley Publishing Company.

[Jarke] M. Jarke, Requirements Engineering in the year 2001: On (Virtually) Managing a Changing Reality. *Workshop on System Requirements: Analysis, Management, and Exploitation*, Schloß Dagstuhl, Saarland, Germany, October 4-7, 1994.

[Medina-Morena92] R. Medina-Mora, T. Winogard, R. Flores, The Action Workflow Approach to Workflow Management Technology, *Conference on Computer Supported Cooperative Work*, Nov. 1992, pp. 281-288.

[Mylopoulos80] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis, Telos: A Language for representing knowledge about information systems, *ACM Transaction on Information Systems*, vol. 8, n4, 1990, pp. 325-362.

[Mylopoulos95] Conceptual Modeling for Information Systems Engineering, *International chaire at the University of Namur (Belgium)*, 1995.

[Schael93] T. Schael, B. Zeller, Workflow Management Systems for Financial Services, *Coocs'93*.

[Scheer] A. W. Scheer, M. Nuttgens, Business Process (Re-)Engineering: Architecture, Reference Models and Toolset.

[Seaman] C. B. Seaman, OPT: Organization and Process Together, CASCON, Toronto 93.

[Yu93a] E. Yu, An Organization Modelling Framework for Multi-Perspective Information System Design, *requirements Engineering* 1993 - *Selected Papers*, J. Mylopoulos et al., eds., Tech. Rpt. DKBS-TR-93-2, Dept. Comp. Sci., Univ. of Toronto, July 1993, pp. 66-86.

[Yu93b] E. Yu, J. Mylopoulos, An Actor Dependency Model of Organizational Work - With Application to Business Process Reengineering, *Proc. Conf. Organizational Computing Systems* (COOCS'93), Milpitas, Calif., Nov. 1-4, 1993, pp. 258-268.

[Yu93c] E. Yu, An Organization Modelling Framework for Information Systems Requirements Engineering, *Proc. 3rd Workshop on Info. Tech. and Systems*, (WITS'93), Orlando, Florida, USA, December 4-5, 1993, pp.172-179.

[Yu94] E. Yu, Modelling Strategic Relationships for Process Reengineering, Dept. Comp. Sci., Univ. of Toronto, October 1994.

[Yu94a] E. Yu, J. Mylopoulos, Using Goals, Rules, and Methods To Support Reasoning in Business Process Reengineering, *Proc. 27th Hawaii Int. Conf. System Sciences*, Maui, Hawaii, Jan. 4-7, 1994, vol. IV, pp. 234-243.

[Yu94b] E. Yu, J. Mylopoulos, Understanding « why » in Software process Modelling, Analysis, and Design, *Proc. 16th Int. Conf. Software Engineering*, May 16-21, 1994, Sorrento, Italy, pp. 159-168.

[Yu94c] E. Yu, J. Mylopoulos, From E-R to A-R - Modelling Strategic Actor Relationships for Business Process Reengineering, *Proc. 13th Int. Conf. Entity-Relationship Approach*, December 13-16 1994, Manchester, U.K., to appear.

[Yu94d] E. Yu, J. Mylopoulos, Towards Modelling Strategic Actor Relationships for Information Systems Development - with Exemples from Business Process Reengineering, *Proc. 4th Workshop on Information Technologies and Systems* (WITS'94), Vancouver, B.C., Canada, December 17-18, 1994. A version of this paper was presented at the *Workshop on System Requirements: Analysis, management, and Exploitation*, Schloß Dagstuhl, Saarland, october 4-7, 1994.

Appendix

Declaration of the Librarian agent

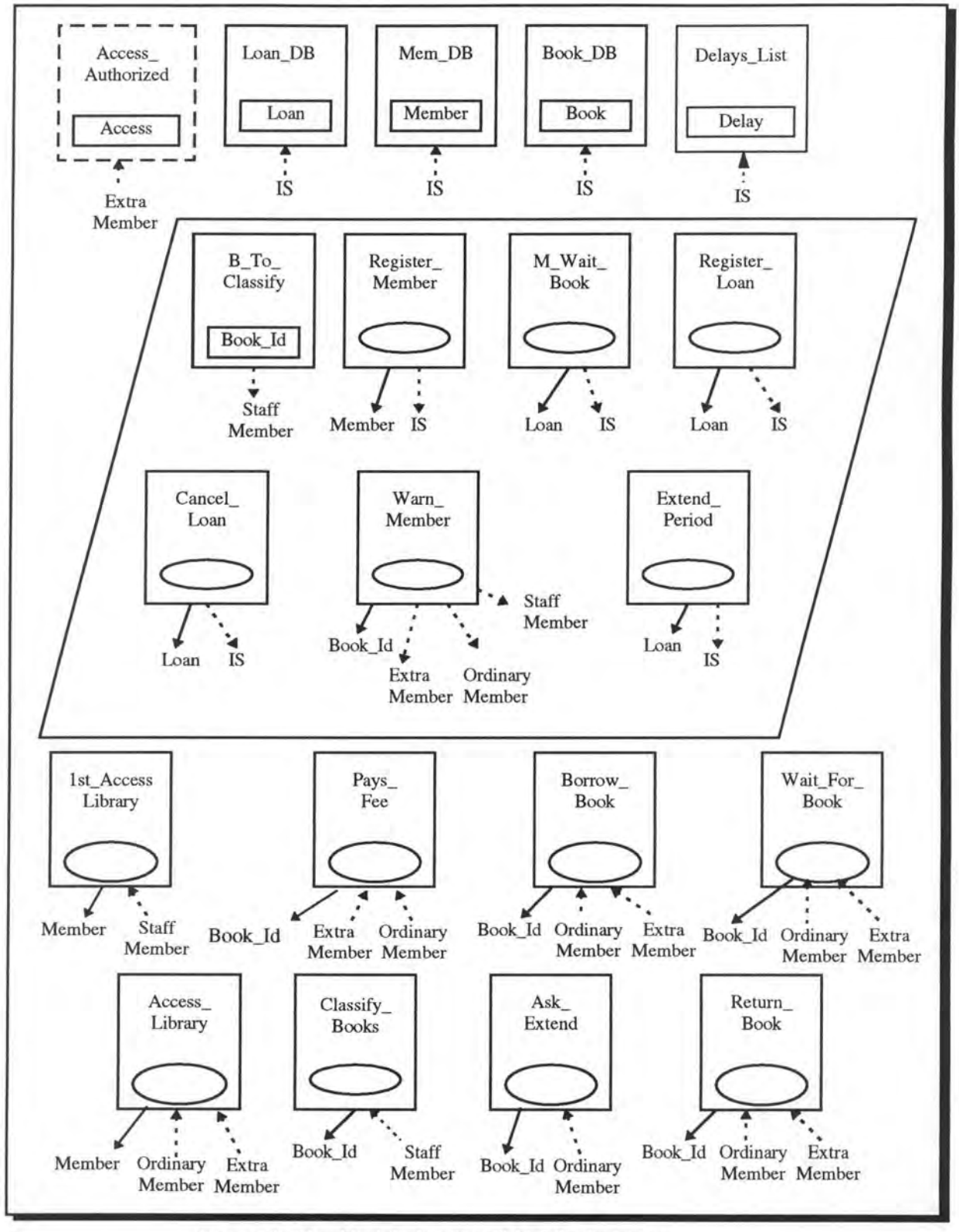
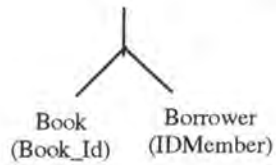


Figure a1 : Graphical declaration of the Libraian agent

Delays_List: the Librarian agent has the list of all the borrowers (Delay) who are late in returning their book.

- The type *Delay* can be described as:



A delay is characterized by the identification of the borrower and of the borrowed book.

Constraints

LIBRARIAN

BASIC CONSTRAINTS

- **Initial valuation**

Card(B_To_Classify) = { }
* *There are no book to classify*

LOCAL CONSTRAINTS

- **Effects of action**

m.Return_Book(b): B_To_Classify = add (B_To_Classify, b)
* *when a member returns a book to the library, the librarian puts it the "B_To_Classify" list*

- **Causality**

Staff Member.1st_Access_Library(m) → Register_Member (Staff Member, m)
* *When a staff member enters the library for the first time, he/she will be registered by the librarian*

m.Borrow_Book(b); m.Pays_Fee(b) → Register_Loan(l)
with Book_B(l) = b
Borrower(l) = m
Waiting(l) = FALSE
(Isof m = ORDINARY MEMBER
⇒ length(l) = 15)

(Isof m = EXTRA MEMBER
⇒ length(l) = m.Time(Access_Authorized)

m.Wait_For_Book(b) → M_Wait_Book(l)
with Book_B(l) = b
Borrower(l) = m
Waiting(l) = TRUE
Length(l) = UNDEF
Date_L(l) = UNDEF

m.Access_Library(member) with ¬ in(Mem_DB, m)
→ Register_Member (m, member)

** If an ordinary member tries to acces the library without being registered, the librarian registers him/her in the member database*

m.Ask_Extend(b) → Extend_Period(l)
with Book_B(l) = b
Borrower(l) = m
Waiting(l) = FALSE
Length(l) = 15

m.Return_Book(b) → Cancel_Loan(l)
with Book_B(l) = b
Borrower(l) = m

- **Capability**

$\mathcal{P}$O(Register_Member(m)/ ¬ in (Mem_DB,m)
$\mathcal{P}$O(M_Wait_Book(l)/ in (Book_DB, Book_B(l))
$\mathcal{P}$O(Warn_Member(b).m/ ∃ d ∈ Delays_List: Book(d) = b
Borrower(d) = m)
$\mathcal{P}$O(Extend_Period(l)/ ¬∃ l' ∈ Loan_DB: l ≠ l'
∧ Book_B(l) = Book_B(l')
∧ Waiting(l') = TRUE)
$\mathcal{P}$O(Cancel_Loan(l)/ in (Loan_DB, l)

COOPERATION CONSTRAINTS

- **Action Perception**

ℵK(m.Classify_Book(b)/TRUE)
ℵK(m.1st_Access_Library/TRUE)
ℵK(m.Access_Library/TRUE)
ℵK(m.Return_Book(b)/TRUE)

- **State Information**

XK(B_To_Classify.m/TRUE)

- **Action Information**

ℵK(Warn_Member(b, bor).m / bor = m)
** the librarian can only warn a borrower how is late in returning a book b*

- **State Perception**

ℵK(m.Access_Authorized/TRUE)
ℵK(IS.Loan_DB/TRUE)
ℵK(IS.Mem_DB/TRUE)
ℵK(IS.Book_DB/TRUE)
ℵK(IS.Delays_List/TRUE)

Declaration of the Staff Member agent

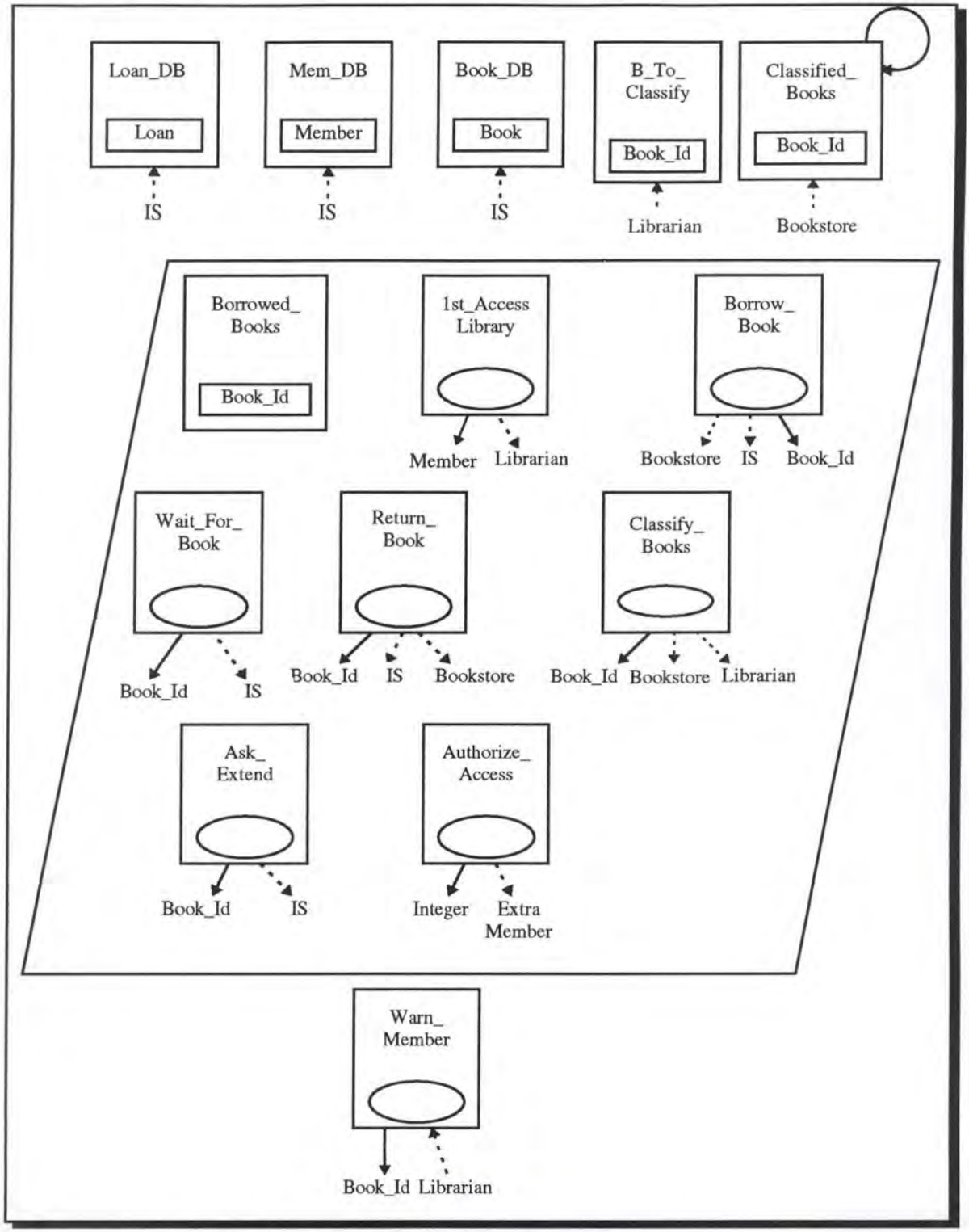


Figure a2 : Graphical declaration of the Staff Member agent

STAFF MEMBER

BASIC CONSTRAINTS

- **Initial valuation**

Card(Borrowed_Books) = { }
* *the staff member has no borrowed books*

LOCAL CONSTRAINTS

- **Effects of action**

Return_Book(b) : Borrowed_Books = remove(Borrowed_Books, b)
Borrow_Book(b) : Borrowed_Books = add(Borrowed_Books, b)

- **Capability**

$\mathcal{P}$O(Return_Book(b)/in (Borrowed_Books, b))
$\mathcal{P}$O(Classify_Books(b)/in (Librarian.B_To_Classify, b))
$\mathcal{P}$O(Borrow_Book(b)/in (Bookstore.Classified_Books, b))
$\mathcal{P}$O(Wait_For_Book(b)/ in (Book_DB, book) with Identification(book) = b
Borrowed(book) = TRUE
$\mathcal{P}$O(Ask_Extend(b)/ (in (Borrowed_Books, b))
$\wedge$ ($\neg$ in (Loan_DB, l)
with Book_B(l) = b
$\wedge$ Waiting(l) = TRUE))

COOPERATION CONSTRAINTS

- **State perception**

$\mathcal{P}$K(Bookstore.Classified_Books / TRUE)
$\mathcal{P}$K(IS.Mem_DB / TRUE)
$\mathcal{P}$K(IS.Book_DB / TRUE)
$\mathcal{P}$K(IS.Loan_DB / TRUE)
$\mathcal{P}$K(Librarian.B_To_Classify / TRUE)

- **Action information**

ℵK(Borrow_Book(b).m / TRUE)
ℵK(Return_Book(b).m / TRUE)
ℵK(1st_Access_Library.Librarian / TRUE)
ℵK(Authorize_Access(i).Em / TRUE)
ℵK(Classify_Books(b).m / TRUE)
ℵK(Ask_Extend(b).IS / TRUE)
ℵK(Wait_For_Book(b).IS / TRUE)

Declaration of the IS agent

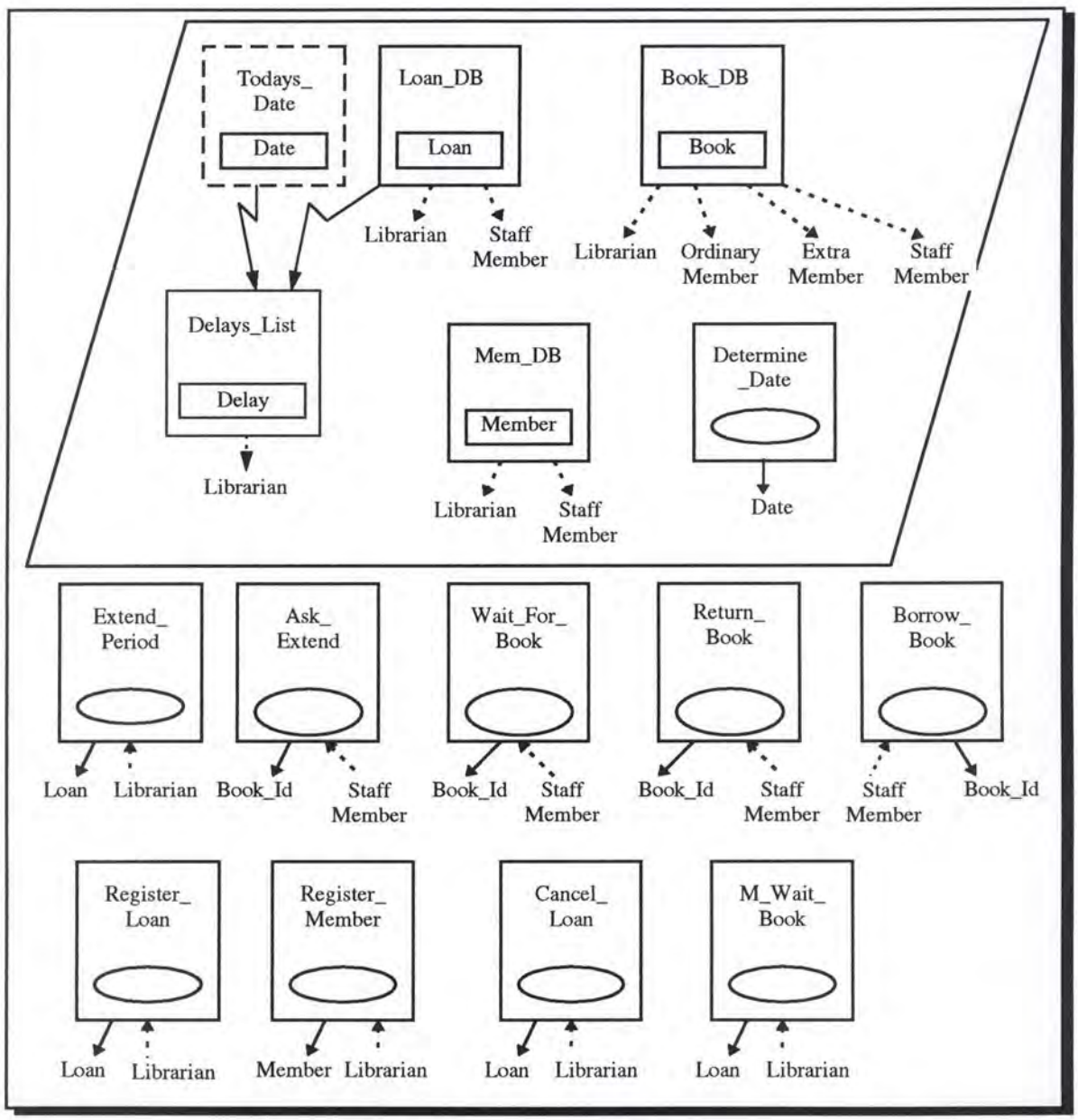


Figure a3 : Graphical declaration of the IS agent

IS

BASIC CONSTRAINTS

- **Derivation rule**

Delays_List = {d}:
$\forall$ d $\in$ Delays_List
$\Rightarrow \exists$ l $\in$ Loans_DB:
Borrower(l) = Borrower(d)
Book_B(l) = Book(d)
Date_L(l) + Length(l) > Todays_Date

- **Initial valuation**

Card(Borrowed_Books) = { }
Loan_DB = { }

LOCAL CONSTRAINTS

- **Effects of action**

Determine_Date(d): Todays_Date = d
sm.Borrow_Book(b):Loan_DB = Add(Loan_DB, l)
with Book_B(l) = b
Borrower(l) = sm
Date_L(l) = Todays_Date
Wainting(l) = FALSE
Length(l) = 15
sm.Return_Book(b): Loan_DB = Remove (Loan_DB, l)
with Book_B(l) = b
Borrower(l) = sm
sm.Wait_For_Book(b): Loan_DB = Add(Loan_DB, l)
with Book_B(l) = b
Borrower(l) = sm
Date_L(l) = Todays_Date
Wainting(l) = TRUE
Length(l) = 15
sm.Ask_Extend(b): Loan_DB = Modify(Loan_DB, l)
with Book_B(l) = b
Borrower(l) = sm
Date_L(l) = Todays_Date
Wainting(l) = FALSE
Length(l) = 15
l.M_Wait_Book(l): Loan_DB = Add(Loan_DB, l)
with Date_L(l) = todays_Date

l.Cancel_Loan(l): Loan_DB = Remove(Loan_DB, l)
l. Register_Member(m): Mem_DB = Add(Mem_DB, m)
l.Register_Loan(l): Loan_DB = Add(Loan_DB, l)
with Date_L(l) = Todays_Date
l.Extend_Period(l): Loan_DB = Modify(Loan_DB, l)
with Date_L(l) = Todays_Date

COOPERATION CONSTRAINTS

State Information

ℵK(Loan_DB.m/TRUE)
ℵK(Book_DB.m/TRUE)
ℵK(Delays_List.Librarian/TRUE)
ℵK(Mem_DB.m/TRUE)

- **Action Perception**

ℵK(l.Extend_Period(l)/TRUE)
ℵK(l.Register_Loan(l)/TRUE)
ℵK(l.Register_Member(m)/TRUE)
ℵK(l.Cancel_Loan(l)/TRUE)
ℵK(l.M_Wait_Book(l)/TRUE)
ℵK(sm.Borrow_Book(b)/TRUE)
ℵK(sm.Return_Book(b)/TRUE)
ℵK(sm.Wait_For_Book(b)/TRUE)
ℵK(sm.Ask_Extend(b)/TRUE)

Declaration of the Ordinary Member agent

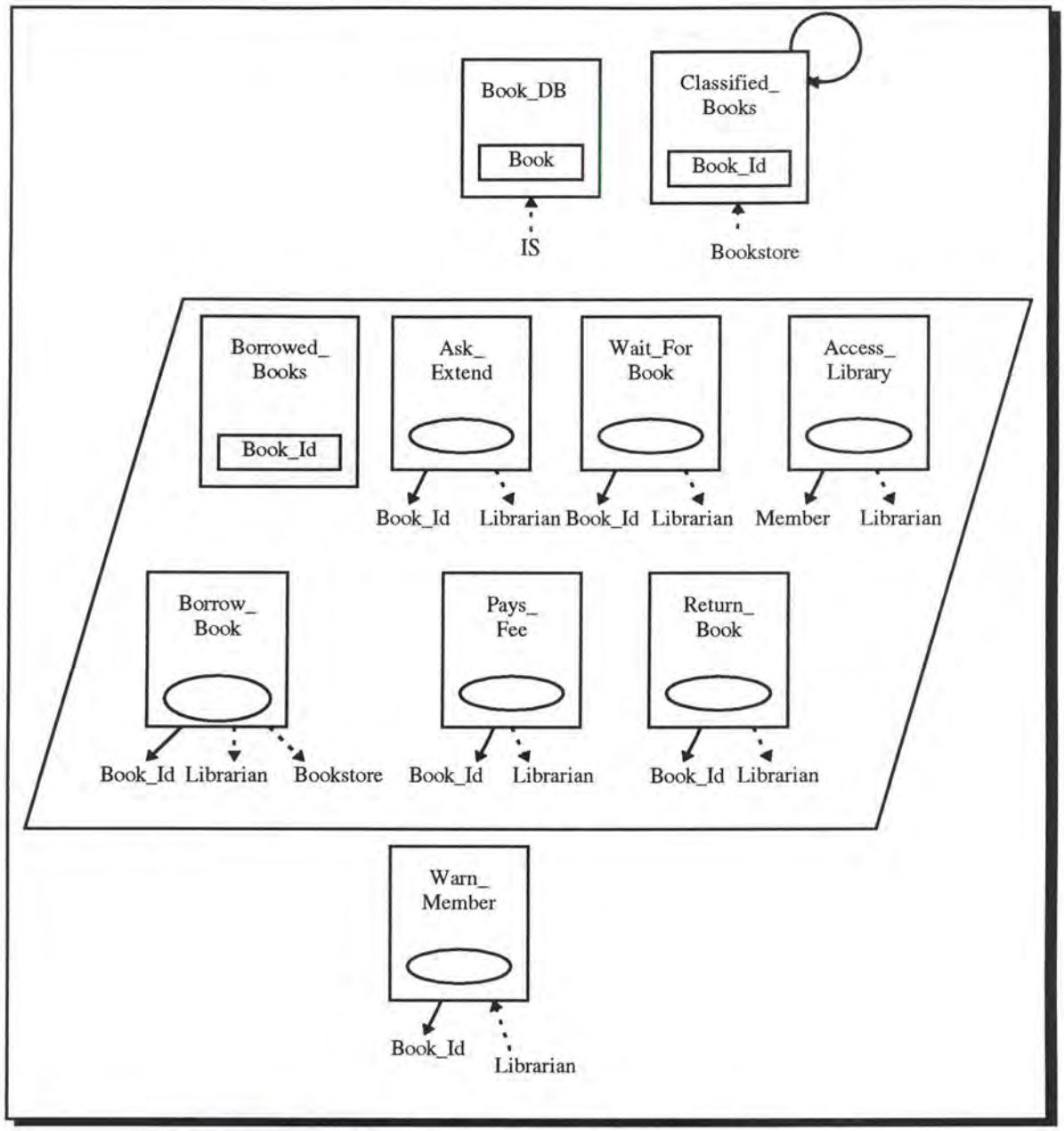


Figure a4 : Graphical declaration of the Ordinary Member agent

ORDINARY MEMBER

BASIC CONSTRAINTS

- **Initial valuation**

Card(Borrowed_Books) = { }

LOCAL CONSTRAINTS

- **State behaviour**

Card(borrowed_Books)≤ 2

- **Effects of action**

Return_Book(b) : Borrowed_Books = remove(Borrowed_Books, b)
Borrow_Book(b) : Borrowed_Books = add(Borrowed_Books, b)

- **Causality**

Borrow_Book(b) → Pays_Fee(b)

- **Capability**

XO(Return_Book(b)/in (Borrowed_Books, b))
XO(Borrow_Book(b)/in (Bookstore.Classified_Books, b))
XO(Ask_Extend(b)/in (Borrowed_Books, b)
F(Wait_For_Book(b)/in (Borrowed_Books, b)
F(Borrow_Book(b)/card(Borrowed_Books) = 2)

COOPERATION CONSTRAINTS

- **State perception**

$\mathcal{K}$K(IS.Book_DB / TRUE)
$\mathcal{K}$K(Bookstore.Classified_Books / TRUE)

- **Action information**

$\mathcal{K}$K(Return_Book(b).Librarian / TRUE)
$\mathcal{K}$K(Access_Library.Librarian / TRUE)
$\mathcal{K}$K(Pays_Fee(b).Librarian / TRUE)

Declaration of the Extra Member agent

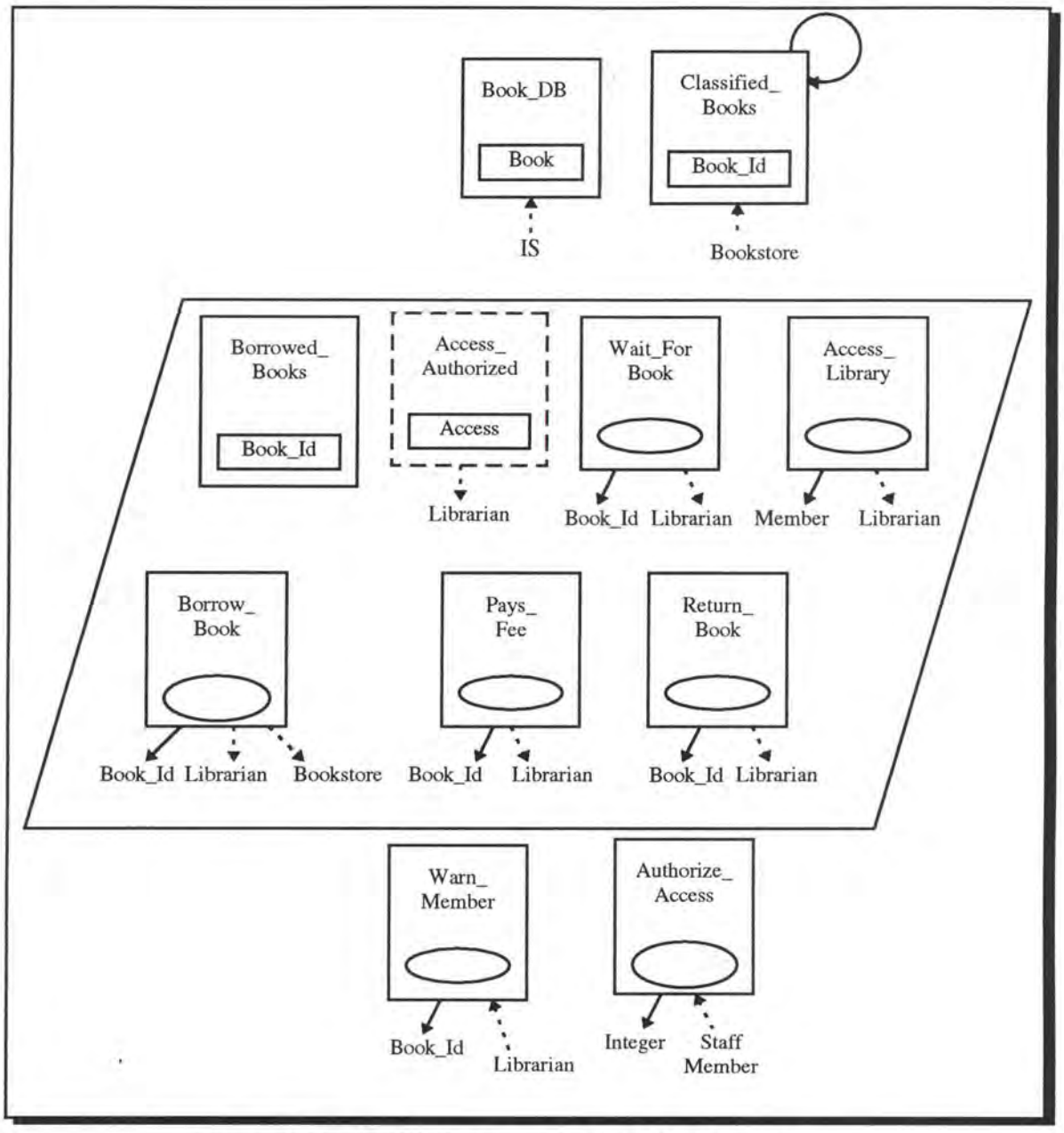


Figure a5 : Graphical declaration of the Extra Member agent

EXTRA MEMBER

BASIC CONSTRAINTS

- **Initial valuation**

Card(Borrowed_Books) = { }
Access_Authorized = FALSE

LOCAL CONSTRAINTS

- **State behaviour**

Card(borrowed_Books)≤ 2

- **Effects of action**

Return_Book(b) : Borrowed_Books = remove(Borrowed_Books, b)
Borrow_Book(b) : Borrowed_Books = add(Borrowed_Books, b)
m.Authorize_Access : Access_Authorized = TRUE

- **Causality**

Borrow_Book(b) → Pays_Fee(b)

- **Capability**

XO(Return_Book(b)/in (Borrowed_Books, b))
XO(Borrow_Book(b)/in (Bookstore.Classified_Books, b))
XO(Ask_Extend(b)/in (Borrowed_Books, b)
F(Wait_For_Book(b)/in (Borrowed_Books, b)
F(Access_Library/Access_Authorized = FALSE)
F(Borrow_Book(b)/card(Borrowed_Books) = 2)

COOPERATION CONSTRAINTS

- **State perception**

$\mathcal{K}$K(IS.Book_DB / TRUE)
$\mathcal{K}$K(Bookstore.Classified_Books / TRUE)

- **Action information**

$\mathcal{K}$K(Return_Book(b) . Librarian / TRUE)
$\mathcal{K}$K(Access_Library . Librarian / TRUE)
$\mathcal{K}$K(Pays_Fee(b) . Librarian / TRUE)

- **State Information**

$\mathcal{K}$K(Access_Authorized.librarian/TRUE)

- **Action Perception**

$\mathcal{K}$K(m.Authorize_Access(i)/TRUE)

Declaration of the Bookstore agent

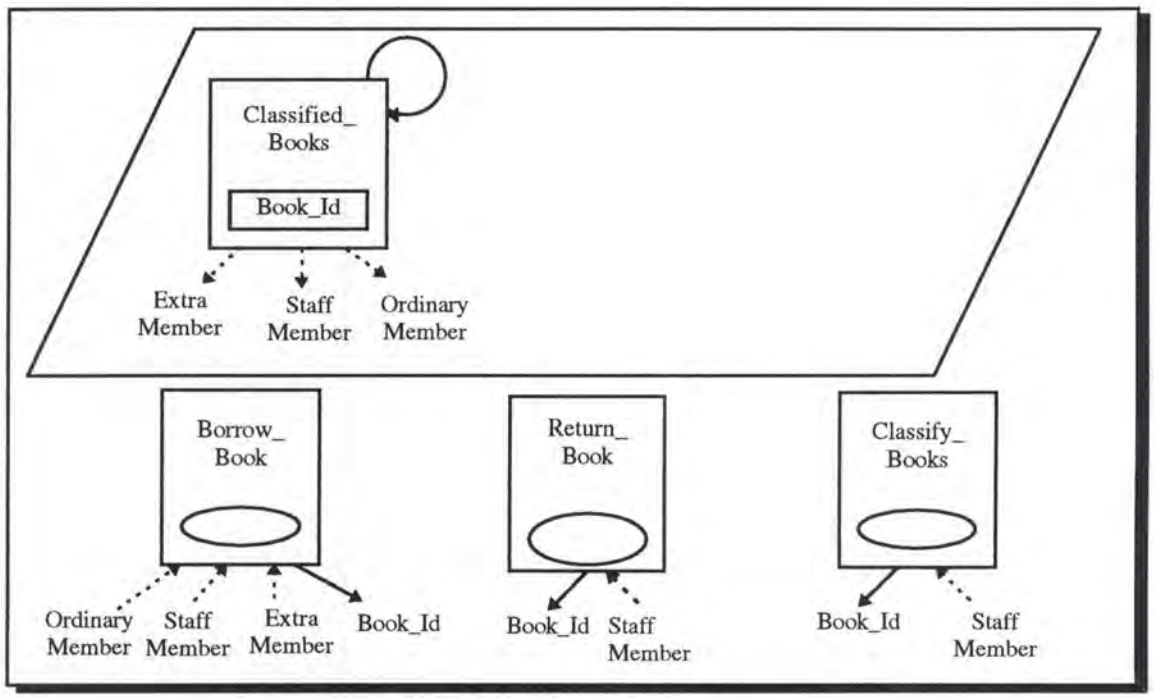


Figure a6: Graphical declaration of the Bookstore agent

Constraints

Bookstore

LOCAL CONSTRAINTS

- **Effects of action**

m.Borrow_Book(b): Classified_Books = Remove(Classified_Books, b)

m.Return_Book(b): Classified_Books = Add(Classified_Books, b)

m.Classify_Books(b): Classified_Books = Add(Classified_Books, b)

COOPERATION CONSTRAINTS

- **Action Perception**

XK(m.Borrow_Book(b)/TRUE)
XK(m.Return_Book(b)/TRUE)
XK(m.Classify_Book(b)/TRUE)

- **State Information**

XK(Classified_Books.m/TRUE)