Abstract.

This paper presents the Hyper-Agenda project whose main aim is to model a system in order to help the organisation and realisation of Tasks which can be applied in an office environment. Two tools that translate the main functionalities of our system are defined:
- the Agenda tool which permits the organisation of Tasks in the Hyper-Agenda environment,
- the Interpretation tool which helps users to achieve their Tasks.

The main concepts of this system are discussed and argued: Entities, Groups, Tasks and Agendas. The key concept of our study is the Task, which allows the modelling of an employees activities within an organisation. Hyper-Agenda allows the user:
1) to express the model of his Tasks;
2) to instantiate these models in order to achieve real Tasks;
3) to plan instances of Tasks into Agendas;
4) to follow their execution. We show that the modelling of Tasks demands tree-like complex objects and that the structuring of entities leads to the point of view notion. Finally, we show that Agendas are derived objects very close to the database view concept. The general expression of constraints and the temporal aspects that allows planning are briefly discussed. A formal specification language to precisely determine the properties of each concept and the functionalities of our system is presented. We conclude the present article giving a brief description of the first prototype developed and outline the future work in this area.

1 Introduction

Office automation is still a formidable challenge: simple functional tools such as word processors are widespread, but Task management systems have only reached a prototype state (DOEIS[1], MAD[2], AMS[3], SOS[4]).

Three types of difficulties explain this fact: the difficulty of stating the system goals, that of a suitable model of Task, and that linked to the underlying technology to be used. Hyper-Agenda is a distributed system for Task management and planning which can be applied in office organisation. This paper introduces the goal, the architecture, the main concepts and the specification of the Hyper-Agenda system.

Hyper-Agenda provides the following answers to the three previously mentioned difficulties:

- the main goal of the system is to help the management of Tasks rather than to automate their execution.

The details of Tasks in all possible cases cannot be described in many applications because exceptions are too numerous. The only automation that will be processed is the insertion of Tasks into an Agenda. In other words, Hyper-Agenda represents knowledge related to Tasks, in order to help users to execute them. Automation might be added as an extra layer above the query and data management system. However, to avoid ambiguities, a formal specification language was defined to determine precisely the properties of each concept and the functions of our system.

- the Task model adopted retains the common features of similar works taking into account our main goal.

To help users, Tasks must be described in the system as much complete as possible. They may be embedded and subject to constraints. They act on data such as documents, and need Resources (material or human) for their execution. Also, the description formalism is close both to the classical object model [16] from a structural point of view, and to the process notion [17] from the behaviour point of view.

- tools used for implementation are distributed advanced databases systems, together with friendly interface tools.

The Agenda tool, for example, gives the privileged users view through which they can query, plan or modify
Tasks. Let us go into some details about the previous choices, the combination of which gives Hyper-Agenda its originality.

The first particularity, help the user rather than do the Task, stem from experience in office work. Office Tasks may be divided into two subsets [5]:

- highly structured Tasks for which a detailed resolution method is known: these Tasks are like programs.

- loosely structured Tasks in the sense that they cannot be described in a sufficient level of detail to be executed by the system, mainly because they depend on the context (i.e, with numerous exceptions). These Tasks demand creativity, initiative and originality. Here the only thing that can reasonably be done is to build systems which help the user to define their goals and their strategies [6, 7]. We aim at building such a system, taking into account the following remarks: If Tasks must be registred within the system but help little, the tool will be discarded. The true challenge here is to build performing interfaces which never bother the user. Lastly we think that only an experimental use of this tool will lead to a relevant automation. Most works focusing on the Task model stop before that phase ([2, 8, 9, 10]).

The second particularity is to stick to a more complete Task model which tries to enforce the process of helping by making the system as much cooperatif as possible. In fact, all the information that a user may need to accomplish his activity must be described in the system using the concept of Task. Close concepts must be unified to avoid jargon, for example, persons and things for which having an Agenda makes sense are gathered into the concept of entities.

The third particularity is to use advanced DBMS features which state that: (1) the knowledge related to Tasks (or organisation) must be managed by a DBMS; (2) this knowledge may be complex and from different types (static, rules, procedural) and, consequently, demands advanced functionalities from this DBMS; (3) a correct functioning must rely on a proper concurrency control mechanism. The Agenda support information that could be shared by different users, and considered mainly as an interface object close to the view notion in DBMS [15]. Special care was taken to stay independent from the application domain. Only experiments in other fields will confirm this claim. Since our paradigm is based on the instantiation of models in order to get executable Tasks, it is clear that our model cannot handle all possible situations, i.e., we cannot improvise a Task first without modelling it, and we cannot dynamically modify the models once they have been used.

A first prototype of Hyper-Agenda has been implemented and will be briefly described in the conclusion of this paper. The paper is organized as follows:

In Section 2 the architecture and the main functionalities of Hyper-Agenda would be defined.

In Section 3 the four main concepts of Hyper-Agenda will be introduced: Task, entity, Group and Agenda.

The fourth Section presents a formal specification language and its application on the concepts and functionalities of Hyper-Agenda. In Section 5 we conclude and present future work.

2. Hyper-Agenda: Architecture and Functions

Figure 1 shows the three components of the Hyper-Agenda system:

1) The Hyper-Agenda environment represents the external level of the system. It is composed of employees who execute activities and the material objects used for the realisation of these activities.

2) The Hyper-Agenda Tools component represent the interface level between employees and the system. It is composed of two tools which translate the main functionalities of the Hyper-Agenda system: the Agenda tool and the Interpretation tool.

- The Agenda tool is built from the Agenda concept and it will be presented in the following sections. This tool allows workers to invoke management operations and organisation of their activities. The main functions of the Agenda tools are:
  - The visualisation of activities through the Agenda.
  - The creation of activities in the Agenda.
  - The temporal planification of activities in the Agenda.
  - The questioning of activities from the Agenda.

- The Interpretation tool allows users to make use of information needed either to achieve their activities or information about activities at run time. The main functions of this tool are:
  - The steps to be followed in achieving activities.
  - The data or objects necessary for the execution of activities.
  - The constraints which may be enforced before, during and after execution of activities.

3) The Hyper-Agenda database allows storage of informations which concern the activities. In particular the tracing execution of activities are stored to know and query their histories. The following section present the main concepts which govern the implementation of the two tools translating the functions of the Hyper-Agenda system.
3 The main concepts of Hyper-Agenda

The main concepts of Hyper-Agenda are: Entity (Actors or Resources which may have an Agenda), Task (activities in the organisation we consider), Group (set of Actors that may realize together one activity), and Agenda (the object which allows the user to visualize and manipulate its activities). This Agenda concept permit to build the Agenda tool presented above. Of course Hyper-Agenda tolerates applications specific objects (such as documents used in office activities) which will be called Application Object. The relationship between these concepts are depicted in Figure 2:

An Actor (or a Group of Actors) may plan and execute a Task in the Hyper-Agenda environment. The planning process is entirely performed by the system, whereas the execution is done by the Actors with help of the system. The system through the Interpretation tool, will inform the Actors about all the information necessary to achieve their Tasks.

Tasks need Resources to be executed and modify the application objects as a result. The Agenda allows to show views about Tasks scheduled to users. These views materialize the life of an Actor (the set of Tasks which it has been involved in) as well as the life of Resource.

Actors may operate on Agendas through specific operations described later.

The paradigm of Class and Instances of object oriented systems is present in all the design of Hyper-Agenda: entities or Tasks will be described by a set of classes which related themselves by an Inheritance relationship (or point of view, discussed later) and a particular entity or Task will be an instance of one of these classes. Agendas are also instances, but of a special kinds: views, which will also be introduced later.

We assume that those concepts are materialized by object classes of a suitable DBMS. As we go further and further, the requested features of this DBMS will be stated.

Figure 3 shows the relationship between the classes associated to the concepts of the model:
3.1 Entities

Actors and Resources have been Grouped into entities. An Entity represents any "object" concerned by Task execution having the following property: *it can only do one activity at a time (exclusivity of activity)*. This implies that the set of Tasks into which any entity is engaged can be totally ordered and constitutes its Agenda. Entities are managed (their lives may be traced down) through their links to Tasks. Properties of entities will be described independently from those of Tasks. Generally, entities have a life duration greater than Tasks. Remember that entities will be described by classes whose instances will be managed.

Another important point is that, in the real world, an entity may belong to several not embedded classes. For example, a room is both a meeting room and a class room; a person may simultaneously be an employee, a customer and a club member. In such a case, we say that the entity has several roles, and we choose to represent this situation by the "point of view" notion described later which is more dynamic and more natural than a multiple inheritance link used in most object oriented systems.

a) Actors.

Actors are the persons whose time is required for achieving Tasks. They are represented as participating in Tasks within Hyper-Agenda. An active Actor, called an agent, will be able to execute a given Task. The representation of Actors in our system will permit to identify persons who could plan or realize one type of activity. All the activities planned and achieved would be stored in the system. Information about these activities and concerning one Actor could be accessed at any time.

From this concept of Actor, the concept of Group may be defined to represent a set of Actors which belong not necessarily to the same class, and are considered as a unique entity. The concept of Group is necessary, to define cooperative Tasks that involve several actors for their realisation. For example: "Meeting", "Writing a paper",... As an example of Group, the team Database systems in a given research laboratory, which may be formed of different Actors (Professor, Lecturer, Phd Student,...).

b) Resources.

Resources are the available facilities that may be used to accomplish a Task, provided that the exclusivity of activity constraint is fulfilled. They do not achieve the Task as the Actors do, but they help in its achievement. In office automation application, they may be printers or meeting rooms.

These two types of objects have been gathered into the entity concept because on many aspects they behave similarly: their activity may be put into Agenda, they may be Grouped functionally in order to play the same role in some Tasks.

Besides entities, other objects with arbitrary structures depending upon the application, may exist: the Application-Object, for example, a bank account, salary rates and documents.

c) The multiple roles of entities.

In Hyper-Agenda real entities are instances of classes. Their last common property is a name that identifies them into their class. Other attributes depend strongly upon the application. In the application we are interested in, entities will play multiple roles. For example, a person may be a teacher, a researcher and at the same time sportsman. Classical inheritance and specialization notions found in Object Oriented DBMS lead to multiple inheritance which is not already the best representation of entities [12]:

An object may have many roles that come and go over time. Rather than, being an instance of some unique subclass defined through multiple inheritance, an object is simply an instance of many types by virtue of having many roles. Every object reference is related to a particular role, and the behaviour of the object depends on which role is being referenced.

The main drawbacks of multiple inheritance are:

- a useless complexity of the inheritance graph in case of hollow classes,
- the creation of classes for Grouping instances but having no meaning,
- the problems of name conflict.
The point of view notion[11] or multiple roles objects [12] avoid these problems (Figure 4). Each object in the system should have a unique identity (the identity link) and may have many representations (the representation links).

![Diagram showing relationships between different roles and links](image)

Figure 4: the mechanism of "point of view" for entities

### 3.2 Tasks

Tasks enable the modelling of the activities of the environment and allow the representation of the dynamic aspect of this environment. Here, the distinction between the Class which describes the schema of a Task and an instance which stands for a given Task is fundamental. In office automation, a general administrator will create schemas whereas a clerk will only instantiate some of them.

At a schema level, the Task concept will be defined in our system by three components:

\[\text{Task-Schema} = (\text{Data}, \text{Constraints}, \text{Body})\]

The Figure 5 shows the three components, their properties and the semantic of each property in the definition of Tasks.

Let us take the example of a Lecture Committee Task for a conference. Submitted papers are distributed among the members of this Committee in order to be analyzed and graded. Later, a general meeting will then decide which papers will be accepted. The schema of such a Task may be represented as follows.

**Task Selecting-papers;**

\[
\begin{align*}
\text{agent} : & \quad \text{Lecture-committee}, \\
\text{Resource} : & \quad \text{Meeting-room,} \\
\text{param-In} : & \quad \text{(Submitted-papers, MinTosub, MinTosel),} \\
\text{param-out} : & \quad \text{(Selected papers, Notification-letters)}
\end{align*}
\]

- **advancement-state**: Created, Planned, Running...
- **average duration**: 1 Month,
- **creation-date**: Date,
- **limit-date**: Date,
- **execution-date**: Date,
- **sub-Tasks**: (Review-papers, Selection-decision, Notification of acceptance)
- **precondition**: Card (submitted-papers) \( \geq \) MinTosub,
- **postcondition**: Card (selected-papers) \( \geq \) MinTosel,
- **body**: SEQ
  - Review-papers (agent\(\rightarrow\)LC),
  - Selection-decision (agent\(\rightarrow\)LC, resource\(\rightarrow\)MR),
  - Notification of acceptance (agent \(\rightarrow\) LC)
  - End-SEQ

Figure 6: the Lecture Committee Task schema

### 3.2.1 The Data Component

In our model, the data component gathers all data necessary for the creation, plan, execution and management of instance of Task. These data are fields defined at the class level (schema of the Task), and specify, in our system, three types of knowledge:

- **Access right and information for Task planning.** An agent has the right to execute a Task (the program committee in our example). The Resources must be reserved during the execution of Tasks (meeting room in our example). At the planning process, all the Resources needed by a Task during its execution will be reserved. Temporal data such as mean duration time must be given for elementary Tasks while the others are computed by the system. They are used when planning.

- **Evolution of Tasks, life-span overview of entities.** The advancement state of a Task (created, planned, running,...) is kept track of within Hyper-Agenda, in order to help the user querying among Tasks. In Section 3.3 we present the evolution of Tasks at the instance level in our system.

- **Abstract data on which Task may operate and produce other abstract data.** They are represented by the Application-Object concept in our system, and may be associated to the properties param-in and param-out.
The three component of the Task

<table>
<thead>
<tr>
<th>DATA</th>
<th>CONSTRAINT</th>
<th>BODY</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent</td>
<td>precondition</td>
<td>body:</td>
</tr>
<tr>
<td>Resource</td>
<td>postcondition</td>
<td>Action</td>
</tr>
<tr>
<td>param-in</td>
<td></td>
<td>or Structure</td>
</tr>
<tr>
<td>param-out</td>
<td></td>
<td></td>
</tr>
<tr>
<td>average-duration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>creation-date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>limit-date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>execution-date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>advancement-state</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub-Tasks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>average-duration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>creation-date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>limit-date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>execution-date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>advancement-state</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub-Tasks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- who create, plan, and execute a Task.</td>
<td>- constraints on input data of the Task</td>
<td></td>
</tr>
<tr>
<td>- the necessary material objects.</td>
<td>- constraints on output data of the Tasks</td>
<td></td>
</tr>
<tr>
<td>- data on which the Task operates.</td>
<td>- the operational level of the Task</td>
<td></td>
</tr>
<tr>
<td>- data produced by the execution of the Task</td>
<td>If body = Action : the Task is simple</td>
<td></td>
</tr>
<tr>
<td>- average duration of the Task.</td>
<td>If body = Structure : the Task is composite</td>
<td></td>
</tr>
<tr>
<td>- the creation date of the Task</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- limit date of Task execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- date of the Task execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- the advancement state of the Task</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- the sub Tasks of a composite Task</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: the three components of a Task

3.2.2 The Constraint Component

The constraint represent conditions which must be fulfilled by the Tasks. These cover Precondition used before the execution of a Task, and Postcondition used at the end of its execution. Constraints may be customized at the instance level, by adding specific constraints to the general ones expressed in the class definition. We note that, the proper expression of constraints is a very important step in Task modelling.

3.2.3 The Body Component

The body component describes how the Task can be executed. Simple Tasks where the body is represented by an Action. For a simple Task to be performed manually in our system, an action will consist of a textual description. For a simple Task to be performed automatically, it will contain a program call. These simple Tasks may be combined into bigger ones complex Tasks by means of logical-temporal operators in a tree like manner. The body of a complex Task is represented by a structure. Thus, a simplified grammar for the body definition of Tasks in our system is:

```
Body ::= Action | Structure
Action ::= textual-description | program-call
Structure ::= Operator Task-list
Operator ::= SEQ | PAR | REPEAT
Task-list ::= Task | Task Task-list
Task ::= simple-Task | complex-Task
```

3.3 The Instance Level of Tasks

The Tasks at the instance level and after their execution would be represented by all the properties of the Task Data Component which are instantiated during the different stages of a Task instance. The classical notion of "class instantiation" in object oriented languages is translated in our system by one of the following operations:

- **Create-Task**: this operation represents an intention for the execution of a Task by one Actor (or a Group of Actors) in the future. This operation makes the Task instance at the created state.

- **Plan-Task**: this operation represents a confirmation by the Actor of the execution of a Task in the future. This operation makes the Task instance at the planned state.

- **Execute-Task**: this operation starts the execution of a Task which could be accomplished by one Actor (or a Group of Actors) in our system. This operation makes the Task instance at the running state.

Thus, an instance of Task may progress in our system and transit by different states. The Figure 7 shows these different states.
3.4 Agendas

The concept of Agenda permits to build the Agenda tool presented in section 2, that allows workers to invoke operations of visualisation and organisations of their activities. The Agenda concept is defined by three components in our system:

Agenda = (ViewTask, Presentation, Interaction)

The Figure 8 below shows the three components.

The component Presentation concerns the interface part of the Agenda concept (how to present the selected Tasks to the user). The form of posting as well as the set of visible informations in the screen are specified in this component.

The component Interaction specifies the interactive aspect of one presentation. All the operations that an Actor may invoke on a presentation are defined in this component.

One of the originality of our system stem from the way ViewTask components are implemented: they are considered as views in the Relational DBMS sense. Indeed, there are sets of Tasks instances selected upon their concerning a given entity.

Thus, the expression ViewTask ( 'Edson') will give us all the Tasks where the Actor 'Edson' is involved. Note that, the proper use of parameterization will give the component ViewTask of Agendas, the status of generic definition, for both Actors and Resources. For example, the following query:

Create View ViewTask (E : Entity) As
Select * From Task t
Where ( E In t.agent) Or ( E In t.resource)

Can be applied to every entity E in the system. Note that these definitions of the component ViewTask are very close to the virtual class notion [13].
The "purest" implementation will allow Agendas to evolve automatically as soon as some Tasks presented in the selection are changing.

The use of criteria to dynamically select sets of object instances permits to mix the components ViewTask of Agendas in a graceful manner; it is equally simple to get the following Agendas corresponding to the set of Tasks:
- executed by agents X and Y and using Resource R,
- classified as teaching Task or research Task by an appropriate selection condition.

4. Specification of the Hyper-Agenda Concepts and Functions

We have informally presented the concepts and functions of Hyper-Agenda. In this section, we are going to describe briefly the basic notions of our specification language[18] that allows in a precise manner the establishment of all the concepts and functions as regards to our system. It is worthwhile mentioning that the language presented here was inspired by the work done by Cardelli on typed functional languages [19].

4.1 Type, Value and Property

The two basic notions of our languages are: the type and the value. One value represents one occurrence of one type (to represent, respectively, the generic and specific concepts). Among predefined types of our language, we distinguish the basic types (string, integer, . . . ), the temporal types (date, duration, period, . . . ) to plan and organize Tasks into Agendas, and finally constructors (set, list, ∩, ∪, . . .).

The other important aspect of our language is the notion of property which defines a type. In fact, one type will be defined by its name followed by the list of its properties. By property, we understand either one structured property of a given type (like attribute) or a behavioural property acting on the corresponding type (like methods). Each property will be defined by its name and the signature of its arguments. Thus, in our language, the definition of property related to a type will be similar to the specification of an interface in programming language. The following example explains the definition of type Task with two of its properties namely agent and param-in.

Type Task
Property agent : Task → Actor θ Group
param-in : Task → Application-Object Set

The first property states that each Task can not be realized only by an Actor or a Group of Actors. The second property states that the input Task parameter forms a list of application objects. We would like to recall that the aim of our specification language is to define, in a precise manner, the semantics of each concept and functions of our system. Each concept of Hyper-Agenda will be represented by a type whose semantic is given by its properties. Each property will be defined by constraints that establish precisely its semantic. We distinguish, in our language, three types of constraints:

The first two are specific to one property related to a type and will be designed by: Precondition and Postcondition. This two constraints will be generally associated to the properties representing operations. They allow to specify, respectively, the constraints that should be verified before the execution of the operation and after its execution.

The last constraint, which is more general, allows to define the invariants of properties of one type (designed by Invariant) as well as the properties of different types. In the following section, we present the three types of constraints.

4.2 Precondition

For one property P defined on the types A and B (P : A → B), we write as:

Precondition For b = P^A (a) : Φ(a)

To express that the property P may not be defined unless the expression Φ is verified. Φ is one logical expression having one free variable a. As an example, we consider the planning property which allows to insert a Task t in an Agenda of an Actor a in an interval i. Let:

Property planning :
Task x Actor x Interval → Task

Precondition For Planning (t, a, i) :
[canPlan (a, t) ] and [ duration(t) < duration(i) ]

The precondition associated to this property consists of two constraint: the first one verifies if the Actor a can plan the Task t (he must be an agent for the Task), and the second one specifies that the temporal interval i is sufficiently big, so that t could be inserted.
4.2 Postcondition

A postcondition specifies constraints on values of the output type of one property. Thus, for one property \( P : A \rightarrow B \), we write as:

Postcondition For \( b = P_R^A(a) : \Phi(b) \)

To specify that the expression \( \Phi(b) \) is the necessary condition for that the equality \( b = P(a) \), be acceptable. We can express a postcondition on the planning property, defined before, as follow:

Postcondition For Planning \((t, a, i) : [t \in \text{ViewTask}(a) \] and [ interval(t) \subset i ]

This postcondition states that, after the planning operation of a Task \( t \) for an Actor \( a \) in a temporal interval \( i \), the Task \( t \) must be in the component ViewTask of the Agenda of the Actor \( a \) during the interval \( i \). Thus, the Actor \( a \) will execute the Task \( i \) during the interval \( i \).

4.3 Invariant

An invariant will be generally defined by first order logic formulas, and enforces general constraints on one type of property, or property of different types. The following example uses an invariant to precise the constraint of exclusivity of activity (cf. 3.1) for Actors through the property agent of one Task:

- each Actor can only do one activity at a time.

Its specification is:

Property agent : Task \( \rightarrow \) Actor

Invariant \( \forall a : \text{Actor}, \forall t : \text{Task}, [\text{agent}(t) = a] \) and

[Advancement-state(t) = 'planned']

\( \Rightarrow \neg [\bigcup t' \text{ agent}(t') = a \text{ and } t' \neq t) \text{ and } (\text{interval}(t) \cap \text{interval}(t') \neq \emptyset)]\)

With the two notions of precondition and postcondition which defines constraints on one property of one type, the invariant notion enables the specification of types in our language.

Our goal here is to use the specification language for states constraints which give meaning to the concepts and functions of our Hyper-Agenda system. Thus, the semantic of each concept and function is defined without ambiguity. The constraints defined are minimal, in the sense that they don't take into account the structural detail of concepts: only their existence and a minimum of properties that they verify are stated. We think that, this phase of specification is very important to state the system goals and simplify the passage to the implementation phase of our system.

5. Conclusion and Future Work

The work presented here is part of the Hyper-Agenda project whose aim is to model a system to help the organisation and realisation of Tasks. The main concepts of Hyper-Agenda have been presented: Entity, Task, Group and Agenda. We have shown that they call for advanced features compared to what current database systems offer. Complex structures, constraints, point of view and views are notion permitting a natural expression of these concepts. The improvement of the Task model using the preceding notions, are under study. A formal specification language was defined to determine precisely the properties of each concept and the functions of the Hyper-Agenda system.

A first prototype of Hyper-Agenda has been implemented. It runs on a macintosh and has been written in HyperTalk, the underlying languages of HyperCard system. It offers some services from the Agenda tool (visualisation, questioning and planning of simple activities) in HyperCard spirit: robustness and efficiency are ignored in the prototype phase.

Our future work would be focused in the following directions.

- Firstly, an implementation of Hyper-Agenda is being rewritten in Objective-C [20], a compiled language on a next workstation with Sybase as a relational engine. It should be operational by mid-1995. The new version of Hyper-Agenda will be applied to office activities and biology as first application domains.

- Secondly, our work would be centered on the specification and the realisation of the Interpretation tool whose main function is to show to users the steps to be followed for achieving Tasks. In this context, we think that a Petri Nets [21], with some appropriate extensions would be suitable to build our Interpretation tool.

6. Acknowledgement

Financial support of FAPEMA is gratefully acknowledged
References