An office representation framework for planning and monitoring office activities

Alain Michard

INRIA

UNITE DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 891

AN OFFICE REPRESENTATION FRAMEWORK FOR PLANNING AND MONITORING OFFICE ACTIVITIES

Alain MICHARD

AOUT 1988

RR_0891

An Office Representation Framework for Planning and Monitoring Office Activities

Une Représentation des Objets et Activités de Bureau pour la Planification et le Suivi d'Exécution

Alain Michard

I.N.R.I.A.
Centre de Sophia Antipolis
Route des Lucioles
06560 Valbonne - FRANCE

August 16, 1988

Abstract

We propose an office representation framework allowing multi-agent hierarchical planning and monitoring of complex activities. It is a multi-level representation in which the various office entities and procedures can be represented at various levels of abstraction. It is a generic representation: the abstract entities can be specialised to match exactly with the specificity of a given organisation. And it is -hopefully- a "natural" representation as its design is directed by searching each time it is possible a direct mapping between real-world objects, constraints and tasks, and the corresponding entities in the computer system. An experimental methodology is proposed to give an operationnal content to this naturalness characteristic.

Résumé

Nous proposons un canevas général de représentation des objets et activités de bureau destiné à supporter la planification hiérarchique multi-agents et le suivi d'exécution de tâches complexes. La représentation est hiérarchique, les différentes entités pouvant être représentées à différents niveaux d'abstraction. Elle est génerale, ces mêmes entités pouvant être spécialisées pour prendre en compte les caractéristiques spécifiques à une organisation donnée. Elle vise enfin à être "naturelle", en collant d'aussi prés que possible aux représentations familières aux agents de bureau. Une méthodologie est proposée pour donner une réalité empirique à cette qualité de "naturel" de la représentation. Cette représentation sert de base à l'implémentation d'un planificateur de tâches de bureau qui est brièvement décrit.

1 INTRODUCTION

Various paradigms for representing activity within working organisations have been proposed in the literature. Bracchi and Pernici [Brachi 83] propose to classify them along four general orientations:

- In the data-based orientation focus is put on grouping information together according to semantic characteristics. Information communication is described as transmission of forms. An example of a data-oriented model is OBE [Zloof 82].

- Process-centered representations have been first proposed as a basis for automatic project-management and scheduling techniques. Recently, more general models have been proposed to support genuine plan generation by constraint satisfaction. Examples of such models are the AMS [Tueni 87] and the representation of activity for project management in the Callisto Project [Sathi 85]. Description of office activities as hierarchical plans [Sebillotte 88] can be seen as belonging to this category.

- In Agent-based models, the working organisation is described as a set of functions associated with each agent, and a relation network between agents. These models have been mainly proposed and used for sociological analysis of organisations and for communication analysis [Rice 88].

- Mixed-models make use of several entities (processes, agents, informations, etc.) as a basis of the representation. Our work clearly belongs to this category. Other examples are proposed by [Woo 86] and [Brachi 83].

The interest of this categorisation is to reveal through various examples that the choice of the basic entities around which the model is organised, must be directed by the various processes and computations that the model is going to support. In our case we want to model office organisations to support:

Multi-agent planning. Planning is the process in which to reach a given goal, an adequate set of activities are selected and scheduled taking into account their prerequisites and effects, and eventually their time constraints. An initial plan can be modified to optimize the allocation of limited resources to each task. In complex situations, planning

is generally hierarchical, meaning that the problem is first solved by managing very abstract and general representations of the task, this abstract plan being subsequently refined in more and more precise descriptions. In *multi-agent* planning situations, several active agents have to coordinate their activities to reach a common goal. This is typical of office work. As outlined in [Woo 86], systems supporting or simulating cooperative work have to manage with the spreading of information and of documents over multiple logical locations: computers, departments, etc. There are three possible approaches to deal with this repartition of information: the first one is to centralise information, as it is done in the (centralised) data-base approach. Another possiblity suggested by [Woo 86], is to model activities as active actors capable to move through a network of logical workstations, looking for the information they need. A third possibility is to allow the spreading of documents over several logical locations, but to have a centralised representation of these documents in a single knowledge base. A good point of this last approach is that it enables a important simplification in searching information: any information which accessible by the system is represented with its name, type, date, access-rights, etc. and logical or physical location. Of course experiments would be necessary to verify if this approach is realistic for large organisations.

In an office environment, planning capabilities can be useful for simulation of the organisation functioning and for testing new organisation management rules.

Task monitoring. A computer representation of the office can be used also to monitor the execution of complex tasks. This general functionality could be especially useful as an aid to cooperative activity. When monitoring real office activities, planning new tasks is of course possible with the strong limitation that it is impossible to modify the past (and possibly the present). This is an important difference with classical planners for which there is no reference to the real time ("*Now*" does not make sense for them).

Advice-giving systems. The third reason for which we want to build a representation of the office is that we would like to be able to offer to users advice and explanations about the "right" way to do a given task in his organisation. This means that we do not only want to compute and propose action plans, but also to provide all the information that

constitute the rationale of the plan. For instance, if in the preparation of a business trip, an agent has to fill-in a form and send it to Mr X, he would probably appreciate to know if he is newcoming in the organisation, that any business trip has to be authorised by a Departement Director, that Mr Y is his Dept Director, and that Mr Y has delegated this attribution (authorizing business trips) to Mr X.

From these objectives we shall derive some mandatory features for our modelling schema. Some of them are classical characteristics of every world model supporting plan generation: we need to represent critical (or limited) resources and their allocation criteria among the various activities. We need also logical operators to express synchronisation and concurrency between tasks.

A more original feature comes from the fact that an office representation should allow (in our view) tailorability by end-users to match with the local characteristics of their specific office organisation. At a very abstract level it is possible to define an office task with very general properties which remain true through a large set of organisations. For instance in the definition of the "authorising" task, there will always be an "authorising agent", someone asking for the authorisation, a document supporting the transmission of the request, a purpose and a date. At a more precise level of abstraction, the task "authorising a business trip" is of course a special case of the previous one, and as such inherits of its general characteristics, but this task will have probably a lot of other characteristics that can be defined only for a given set of organisation (French public administrations for instance), and a few others completly specific to a given organisation.

If we dont want a model to remain only an intellectual game, but if we wish that software tools can be built from it, the model must be designed in such a way that specific knowledge can be easily added, modified, by the office agents themselve. The basic entities, the way to handle properties and constraints and the relations among entities must be intuitively "understandable" by persons without any knowledge in the field of A.I. representation techniques, but with good expertise in the office procedure domain.

In fact, tailorability of the office representation requires that the model must be understood and manipulated in several different ways: the user must first be able to read (through adequate visualisation conventions) an existing office representation. Many existing entities might be useful as they are, others might be useful with only slight modifications, instanciations of parameters, etc. The user must also be able to create entirely new entities

that were not foreseen by the basic-system designer and to check for the consistency of his newly-created entities with the already existing ones. In all these tailoring tasks (which are in fact closely connected), the user must of course have a good understanding of the semantics of each of the modelling primitives.

Of course close matching between the end-user mental representation of the office and the formal representation embedded in the software system, cannot be obtained just by good intuition of the designer. We shall describe in our conclusion an iterative design process, mixing experimental validation of the model with office-workers, with the "laboratory" design decisions. We shall just say at this point that the office model described hereafter is a first step in the design allowing to bootstrap this iterative process.[1]

As a testbed for our work, we have tried to represent partly the organisation structure and functioning of the research institute we're working in. In our presentation we shall take examples from this representation. An implementation of a planner (named ASTRE) built upon our representation scheme is underway.

2 OFFICE MODEL OVERVIEW

2.1 Representation Layers

It has been proposed elsewhere [Sathi 85] to organise knowledge representation frameworks along five *layers of representation:* the domain, conceptual, epistemological, logical, and implementation layers. We wish to adopt another organisation for our presentation and to distinguish between the following *abstraction levels:* abstract categories (corresponding to Sathi's epistemological layer), conceptual level (including domain specific concepts), visual representation (describing how an office model is presented to a human reader), and implementation level.

2.2 Abstract Categories

Many general features of our representation sheme are common to most frame-and-rule-oriented representation substrates. We shall insist mainly on what differenciates our one from the others, the main point being the way time is represented, and synchronizing constraints are described.

[1] Furthermore, the present document serves as a specification for the implementation of a planning system.

2.2.1 Concepts and Prototypes

Our basic abstract category is the *concept.*[2] Concepts have properties represented as typed and named slots. The following types are defined:

object: A slot of prototype A can "contain" (or refer to) an object b, instance of a prototype B. This means that an object a, instance of A cannot exist until b has not been defined itself.

prototype: If a slot of A refers to B, a can exist even if B has not been instanciated in an object b.

numeric value: A numeric value is an association between a number and a unit. [3]

cardinal number: Integer value without unit.

numeric interval: Defined by two numeric values.

cardinal interval: Defined by two cardinal numbers.

boolean: To express truth values.

date: A date is a point in the temporal continuum: May, 7th, 1988 at 10h30 am is a date.[4] May 7th 1988 is a real-time interval.[5]

real-time interval: Any interval defined by two dates. The 1988 year is a real-time interval. As we do not offer a specific type for "fuzzy dates", ("around the 1st of June"), they will be represented as time intervals. At the conceptual level, conventions will have to be defined to choose interval widths for each kind of fuzzy date that can be found in natural description of activities by office workers.

day-of-the-year: "Christmas day", or "27 of June" are days of the year when they respectivly refer to any Christmas day or to any 27 of June, and

[2] We could have called it a *frame* but we considered that it is important to make a clear distinction between a concept and its specialisations, the prototypes. In our model there is no inheritance between concepts.

[3] At the implementation level we shall distinguish between integer and real values. At the abstract and conceptual levels we do not find any necessity to make this distinction.

[4] For our purpose the one minute precision seems to be sufficient. For a futur version of our representation we think of introducing a variable grain-size for dates.

[5] At the implementation level it is possible to represent dates as numeric values. At the conceptual level dates are specific types, for the purpose of clarity.

not especially to the one in the current year. We implicitly make use of this type in a sentence such as: "*May 1st* is official holiday".

calendar-interval: This is an interval between two "days-of-the-year".

hour-of-the-day: This is an hour defined for any day, or for a set of days. We very often make use of this type in natural communication: "I usually leave my office at *6 pm*" is an exemple.

watch-interval: This type corresponds to a difference between two "hours-of-the-day". Calendar and watch intervals will be useful to define in prototypes constraints for the instanciations of dates that must be done during the planning process.

Paragraph-types: Three paragraph-types are defined: character-string, graphic and table. These types are mainly used to describe document contents.

Action: An action defines all the side-effects of the creation of an object.

Constraint-rule: A constraint-rule describe constraints that have to be satisfied for instance creation. For these two types see later for a more precise definition and examples.

State: This type is used to define boolean variables that can take one of the two values "sleeping" or "active". It is used only for activities.

Priority: A integer between 1 and 5.

symbol: We shall use a symbol to name any terminal entity. A terminal entity is one for which we dont need to give a structural description (there is no concept).

Any slot can "contain" either a single value of a given type, or a list of several values, all of the same type. Three different processes can result in assigning a value to a slot: local assignment, value inheritance, or method-evaluation. In this last case a a typed function is defined for the slot.

For each concept, a tree of more specific concepts can usually be defined: the prototype hierarchy. A link is this tree of prototypes is generally refered to as an "is-a" link. Going down this tree of prototypes along "is-a" links, we shall define more and more specialised description of the concept just by narrowing the possible values of slots. A prototype inherits properties

from other prototypes along a given branch of the tree: for a prototype the domain of values of one of its slots must be consistent with the value-domain defined higher in the hierarchy for the same slot. Of course, instanciating (i.e.; giving values to each property of the prototype) a prototype results in creating an *object.*

Tasks will be represented as concepts and prototypes. From now on we will call *activity* an instanciation of a task: *activities* are *objects.* Any work procedure in the office will be represented as a task. The realisation of some work will be represented as an instanciation of some task(s), creating activities.

As time operators are defined only between tasks, we shall present them in the conceptual-level section, where task structure is described.

2.2.2 Relations

An arbitrary number of relations can be defined among any subset of concepts or prototypes. Relations can be of any type: transitive or not, reflexive or not, symetric or not.

2.2.3 State

A *state* of the world is completly related to a given set of existing *objects*, and a given set of relations among them. Some properties of these objects can have specified values, others can be constrained to an interval, and some others can be free (*nil* value for this slot). A *goal* is only a particular state, which is generally different from the current state. It's worth noting that as activities are represented as objects, a state contains information about past and present activities.

2.2.4 Rules

A rule is attached to a concept (or to a prototype). It gives indication of constraints that must be satisfied to create or modify an object. Premises of rules refer to the existence of a state. Conclusion of a rule gives indications on a possible modification of the current-state to create a new one. As we shall see, rules are frequently used to simplify the concept-based representation of the world.

2.2.5 Contexts

A context is a set of *tasks* with temporal relationships among them. This notion is different from the one of *group* in GEM [Lansky 86] in which there is a *causal* relationship between activities. We shall give details about the use of contexts in the section devoted to the planning process.

Contexts are represented as concepts.

2.2.6 Events

An event can be represented as a triplet associating a logical proposition, a truth value and a date. In most cases, the logical proposition will define a modification of the current state. Beginning and completion of tasks are events. At the conceptual level two kinds of events will be defined: agent-generated events and outside-world generated events.

2.3 Conceptual Layer

Our main concepts are *tasks, agents, institutions, documents, resources and tools.* Several more domain-specific concepts are necessary to represent a real office organisation. Some of them will be described in a latter section.

2.3.1 Tasks

General Organisation of Tasks Figure 1 is a textual representation of the task *authorizing-a-business-trip.*

It is a sub-task of *Preparing-business-trip*, which is itself a sub-task of *Business-trip. Authorizing-a-business-trip* is a special case (a prototype) of the concept *authorizing.*[6] All tasks are organized along these two orthogonal hierarchies: the abstraction tree defined by the "is-a" links and the sub-tasking tree defined by the "part-off" relation. Figure 2 shows how the task *Business-trip* is decomposed in sub-parts in our representation example. Here are some precisions about some properties of the task *Authorizing-a-business-trip* (we have added the description of some properties which can be found in other task descriptions):

Sub-parts The task is in fact the grouping of two tasks: asking for and giving an authorization. It would have been possible to put directly these

[6] We have chosen an example which have been extensively described in [Tueni 87] for AMS. Readers can note the close relationship between this system and our work.

Prototype: *Authorizing-business-trip* from Concept: *Authorizing*

Part-off:
Type: Proto
Link-type: part-off
Reverse-link: sub-part
Proto-name: *Preparing-business-trip*

Sub-parts:
Type: Proto-list
Link-type: sub-part
Reverse-link: part-off
Domain: *(Ask-bt-authorization, Give-bt-authorization)*

Motive:
Type: Proto
Proto-name: *Business-trip*

Actor:
Type: Object
Proto-name: *Agent*

Involved-agents:
Type: Object-list
Proto-name: *Agent*

Document-support:
Type: Proto
Proto-name: *Bt-form*

.....

End-date:
Type: Date
Value: nil

Constraints:
Type: Rule-list
Domain: *(bt-notice-rule, bt-delegation-rule, ...)*

Side-effects:
Type: Action-list
Domain: *(bt-form-move-2, budget-res-for-bt, ...)*

State:
Type: State
Value: nil

Figure 1: Textual representation of a task

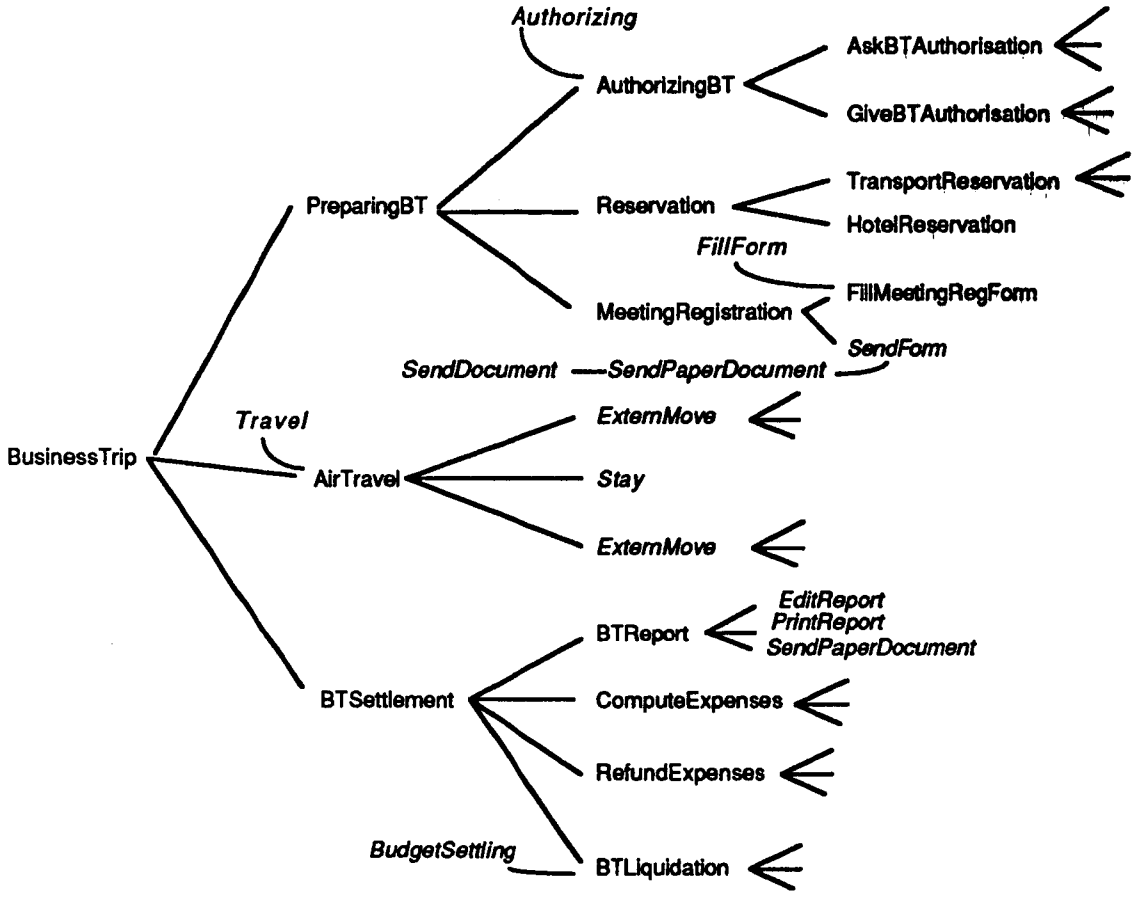

Figure 2: A part of the hierarchy of sub-tasks for *Business-trip*

two subtasks as siblings of "*Preparing-business-trip*". But they share a lot of common properties: involved agents are the same, motive is unique, document supporting and produced by the tasks is the same form, and there are strong time-constraints between the two tasks. Grouping them in an "ad-hoc" higher-level task is convenient but not mandatory. We feel that it is "natural", meaning that this group can be easily understood by office-agents.

As any other property, a sub-part relation is optional.

Actor The "main actor" of the task, responsible of its realisation. This could be for instance the secretary of the person leaving for the business-trip.

Involved-agents There are at least two "involved-agents" who are the two main actors of the sub-tasks.

Document-support The document used to ask for the authorization and to sign to authorize the trip. In most organisations it will be a paper form.

Document-produce For this task, it is the same as the previous one.

Begin-date For this task the begining date will be equal to the begining date of the sub-task "Ask-bt-authorization" (in fact of the sub-sub-task "Transmit-bt-author-form").

End-date It will be equal to the ending-date of the sub-task "Give-bt-authorization".

Average-duration This property is not relevant for this task. When it exists, it is either of type calendar-interval or of type watch-interval. It gives an indication on the average duration that is necessary to perform the given task. In somes cases it can be replaced by an Average-duration-interval. In this case there is generally a rule specifying more precisely the average-duration according to the tool(s) selected to perform the task.

Constraints This slot contains a list of rules that have to be satisfied to create an object instance of the task. Some rules express time-constraints, other ones express administrative constraints. Here is an example of a constraint-rule attached to the task "Give-bt-authorization":

If the *Actor* of "*Business-trip*" is a researcher
and that the *Goal* of the "*Business-trip*" is to attend to a scientific conference,
and that the *Destination* of the *Business-trip* is in a foreign country,
and that the Actor has an accepted Communication at that Conference,
then *Give-bt-authorisation* can be achieved.

Side-effects This is a list of actions that are performed whenever an instance of the task (an activity) is created. Actions can be modifications of properties of other existing objects, or creation of new objects (possibly of new activities).

State State can take one of the two values "*active*" or "*sleeping*", only when an object (activity) is defined. An activity is active only if one of its sub-part is running. Running activity is defined in the section devoted to time operators.

To represent a typical office organisation, we need to define a lot of tasks at various abstraction levels. The most general tasks can be used for any organisation. At this level we can define tasks as: Authorising, Negociating, Going-to, Assisting-a-meeting, Buying, Selling, Recording, Copying, Editing, Filling-a-form, Make-appointment, etc.

At a less abstract level we shall define tasks that can probably be used for many organisation descriptions with only slight modifications. This will be probably the case for Business-trip, Planning-a-budget, Assisting-a-course, Install-workstation, and many others. Then to complete the representation, very specific tasks must be defined. Their structure will probably have to be deeply modified to match to the real work in the various organisations. A good example, that we have described in our representation example is the task "*Draft-annual-activity-report*". Of course the same task shall probably exist in many organisations but its realisation can be completly different from one case to another.

The task-structure description is not sufficient to understand how activities are created, synchronised and completed: it is necessary to give details about the way prerequisites and effects of activities are represented and about the representation and process of time constraints. Before this, we wish to present rapidly a few other concepts, to give the reader a better feeling about our "static" world representation.

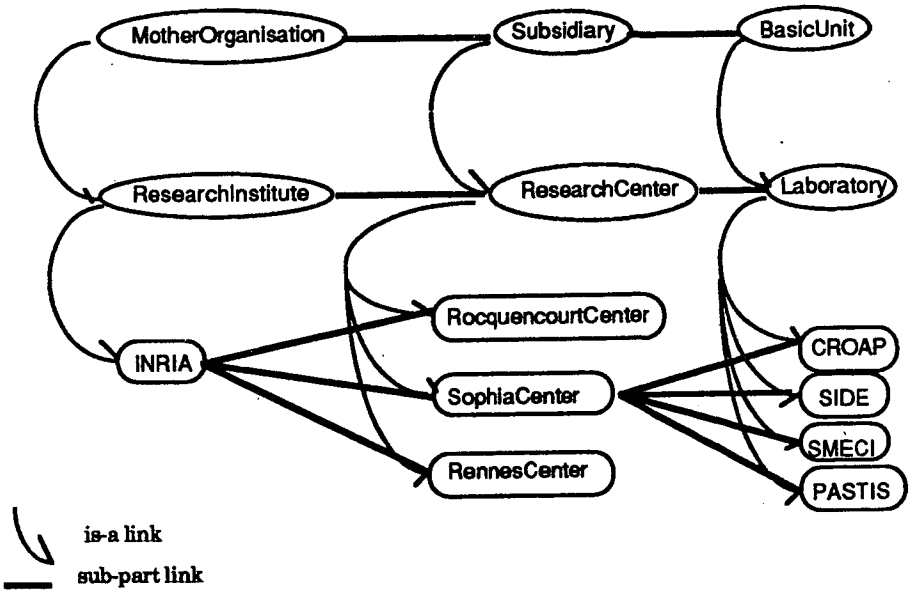

Figure 3: Hierarchy of Managements Units

2.3.2 Agents

The concept describing Agents has the following properties: attributions (they can be permanent or modified by delegations), posting (a management unit to which he is appointed), and name. A hierarchy of prototypes ca be defined for a given institution. For instance, to represent a research organisation, we'll need prototypes for directors, researchers (of various status), administrative agents, and so on.

2.3.3 Management Units

A management unit has depending-units (a list of other management units), a location (a town for instance), a Director (an Agent), and a budget. Other properties could be added for tailoring for specific organisations. Figure 3 shows the hierarchical structure of the various management-unit hierarchy describing a research organisation.

2.3.4 Tools and Resources

The most important tools that have to be represented are communication tools (telephone, fax, audio-conferencing facilities, electronic-mail, etc.), editing tools (text editors, graphic editors, spreadsheet programs), printers and photocopiers. Tools have a location, a cost-of-use, and can be single-user or multi-user. The main idea in the representation is to be able to link rules (constraints) with tools, defining which of them can be used to perform a given activity, and which one are available to a given agent at a given moment.

An important limited resource is money. It is always represented as movements on a *budget* which is currently our only concept for money representation.

Constraints on tools and resources The use of tools and resources for activity creation is directed by constraints that are directly linked to the corresponding prototypes. The most noticeable constraints bear on access-rights (who has the right to use a tool or to consume a resource), on exclusive access on individual tools (with priority), on the mean-duration which is necessary to perform a given activity with a given tool.

2.3.5 Documents

Living Documents "Document" is one of our most important concepts for office representation. In fact nearly all information communication that we can wish to represent can be represented as document transmission. Decisions and negociations also, in most cases,require modifying or creating documents. The fact that many tasks can create, modify or transmit documents means that documents are not static objects: they are born, evolve, move and at the end are "frozen" (archived, published or destroyed). Documents, in a broad sense, are not only a support for or a product of activity. The state of a set of documents can be directly linked to a state of the world: if a document exists, if it has been partly completed, if it is at a given location on a given support (electronic, hand-written or printed paper), this generally means that some office activities have been performed, and that some others have not. Even when an activity does not directly make use or produce written material, a thorough analysis can often reveal a document modification as a side-effect. For instance, making an hotel reservation by telephone call does not (possibly ?) directly produce a document, but if

the task succeeds, some information will be added on some document that exists for another purpose (a note-book, a business-trip form,..). [7]

Document Content, Structure and Layout. To have easy-access to a representation of the current-state of a set of documents, it is necessary to distinguish between *structure, content, and lay-out.* [8] In our office representation we have not adressed the problem of representing the lay-out of a document (neither at the conceptual nor at the implementation levels). All document prototypes have as many slots as necessary to describe their structure. For instance the *Technical-report* prototype has slots for title, author(s), abstract, lists of sections, of subsections and of paragraphs (with sub-parting relations). Paragraphs are the leafs of the structure. They have a type (character-string, graphic, table). The form-prototype has mainly a slot for its list-of-fields. A form-field is a concept that has a label and a type (numeric, interval, date, character-string).

All documents have also slots for author(s), creation-date, last modification-date, access-rights, state (any combination of electronic, printed, hand-written, archived), and for a list of rules.

Document constraints Activities can eventually put or erase some content in the document structure, according to the rules attached to the document. As there are generally a large number of constraints attached to a document, the inheritance of properties (and among them of constraints) in the hierarchy of document prototypes is particulary useful.

Constraints can bear on:

- the order for filling out the structure: in a letter, the signature cannot exist before a header and at least a paragraph has been filled in.
- mandatory states: a technical report cannot be published if it is in the hand-written state.
- on adequation between a state and the use of a tool: a letter cannot be sent by electronic-mail if it is hand-written.

[7] Of course it is possible to find examples of real office activities without any effect on some document. Our sheme allows the representation of such tasks because constraints can be directly attached to those tasks.

[8] This distinction is evident for anyone familiar with the electronic-publishing domain or with syntax-directed editing.

- on access-rights: some documents can be modified only by their author, others can be modified by the author and by his secretary as well. Documents cannot be created by anybody in the organisation.
- on relations between operations and the filling state of the structure: a letter cannot be sent if it is not signed.

2.3.6 Misceallenous Concepts

Many other concepts are necessary to represent activities in an office organisation. We shall not give here an exhaustive description of all of them. Three of them -at least- are important and general (they will be probably needed for any organisation).

The first one is the concept of *Meeting*. A meeting has of course participants, a location, begining and ending dates, and a few other more specific properties at the prototype level. In our example we have defined prototypes for scientific Conferences, for internal seminars, for courses, for the board of directors meeting, for scientific steering-comittee meeting, etc.

Another general concept is the *Budget*. A budget has outlay and receipt lines and a balance. It is linked to a *Management Unit*, and has related constraints for negative balance limitation and access-rights. It is clear that to be able to run real-size experiments, we shall have to define a more sophisticated model of budget structure and management.

The last concept we want to mention here is the one of *Delegation* describing how an attribution can be delegated, permanently or temporary, from an agent to another. An attribution is the possibility for an agent to create or modify an object or a set of objects. [9]

2.4 Time operators and constraints

We have given an overview of a static representation of an office world. We must now describe how we represent time constraints. We have said that an activity has a begining date and an ending date (they could be equal for very simple activities as *Copy-message* for instance). The begining and the end of an activity are *elementary events*. If α and β are two activities, $\mathcal{B}(\alpha)$ and $\mathcal{E}(\alpha)$ are elementary events denoting the begining and end of the activity α , we shall define the predicates:

[9] It is possible to define access-rights to an object as an attribution attached to an Agent, or as an access-right attached to the object itself. There are here possiblities for redonduncy and for contradiction.

$before(\alpha,\beta) \equiv \mathcal{E}(\alpha) \prec \mathcal{B}(\beta)$
$tequal(\alpha,\beta) \equiv (\mathcal{B}(\alpha) = \mathcal{B}(\beta)) \wedge (\mathcal{E}(\alpha) = \mathcal{E}(\beta))$
$beforeq(\alpha,\beta) \equiv \mathcal{E}(\alpha) \preceq \mathcal{B}(\beta)$
$after(\alpha,\beta) \equiv before(\beta,\alpha)$
$afterq(\alpha,\beta) \equiv beforeq(\beta,\alpha)$
$overlaps(\alpha,\beta) \equiv (\mathcal{B}(\alpha) \prec \mathcal{B}(\beta)) \wedge (\mathcal{E}(\alpha) \prec \mathcal{E}(\beta)) \wedge (\mathcal{B}(\beta) \prec \mathcal{E}(\alpha))$
$during(\alpha,\beta) \equiv (\mathcal{B}(\beta) \prec \mathcal{B}(\alpha)) \wedge (\mathcal{E}(\alpha) \prec \mathcal{E}(\beta))$
$starts(\alpha,\beta) \equiv (\mathcal{B}(\alpha) = \mathcal{B}(\beta)) \wedge (\mathcal{E}(\alpha) \prec \mathcal{E}(\beta))$
$finishes(\alpha,\beta) \equiv (\mathcal{B}(\beta) \prec \mathcal{B}(\alpha)) \wedge (\mathcal{E}(\alpha) = \mathcal{E}(\beta))$

If $\mathcal{N}$ is the variable of type date permanently updated to contain the current time, then we can define the following predicates on activities:

$future(\alpha) \equiv \mathcal{N} \prec \mathcal{B}(\alpha)$
$ended(\alpha) \equiv \mathcal{E}(\alpha) \prec \mathcal{N}$
$running(\alpha) \equiv (\mathcal{B}(\alpha) \prec \mathcal{N}) \wedge (\mathcal{N} \prec \mathcal{E}(\alpha))$

A time-constraint is a logical combination (and/or) of a set of time-predicates. A general constraint results from any combination of time-constraints with constraints on the values of object-properties and on the existance of objects.

A *rule* has a general-constraint as premiss, and a set of actions (see below) as a conclusion.

2.5 Actions

The first kind of action which can appear in a rule is the creation of an activity by instanciation of a task prototype. We have seen how the prereq-uisites to an activity creation can be expressed as a general constraint. This creation has in fact three kinds of effects on our world representation: the new activity exists, values can be given or modified to its own properties, [10] and new objects can be created (causal creation).

The other kind of action is the modification of properties of any existing object. Values can be directly assigned to object properties or intervals can be defined or narrowed. Of course these values must be consistant with types and intervals defined in the prototype of the modified object.

[10] Many values are inherited from the prototype. They can be modified by actions. Whenever the prototype does not have a default value defined for a slot, the slot can be filled by two ways: function evaluation (method) or direct assigment by an action.

2.6 Relations

We have already mentioned the "is-a" relation that can be used to construct a hierarchy of prototypes for any concept, and the sub-parting relation which is used to break up tasks in component parts. This sub-parting relation is also heavily used to define the structure of documents.

Other important relations are defined between tasks and agents (*responsible-for, involved-in*), between agents and management units, between agents (*has-authority-on*), and between tasks and documents (*produce* and *support*).

2.7 Requests

At the conceptual level, a request is a *state*. Of course, at the implementation level, the expression of requests by end-users necessitates a powerful user-interface enabling state description through graphical manipulations. This part of our work is still under design.

3 THE PLANNING PROCESS

3.1 Expansion of abstract tasks

The basic planning process in ASTRE consists in expanding a task and its sub-tasks, taking into account the constraints which are attached either to the tasks themselves or to the various objects (agents, tools,..) which are used by or involved in them. During this process a tree of related activities is built. The tree is organised by the sub-parting relation. At a given level of the tree, nodes are ordered by begining dates. (This means that if X and Y are two sibling activities, to say X is at the left of Y, or X comes before Y or X begins before Y are synonymous). A the end of the expansion process all the activities necessary to perform the given tasks have been created, but it is possible that some of their properties are still undefined. This will be the case each time there is no defined constraint in the prototypes to impose a precise value to the property. As a side-effect of this activity creation, documents have been created and modified, according to their own constraints.

We must note that this task expansion is driven mainly by the sub-parting relations among tasks, but that this relationship can be dynamically modified: a sub-part relation can be controlled by a rule, expressing the conditions in which the relation holds true or not.

3.2 Context-driven planning

Expansion of abstract tasks is not sufficient for planning in an office world. In the office each task cannot be performed independently of some of the other ones. We introduce the concept of *context* to manage with interactions between independant tasks.

Contexts We shall try first to give an intuitive idea of what we call a context. Suppose you wish to go to the movies this evening. This task is quite well-known, and you have a good (mental) representation of its sub-parts and of the associated constraints. Suppose now you want to take advantage of this planned activity to realise in the same evening another one: dining at restaurant. [11] This task is also well-known so that you have a good representation for planning it. For this complex planning activity you'll have to take into account constraints that are inherent to each of the tasks, and specific synchronising constraints to merge efficiently the two activities. Furthermore, if after having built an acceptable plan, you decide to add a new task (fill-up the gas-tank of your car, for instance), this will be possible to do it without having to replan completly the other tasks. In our artificial world we call *context* such a set of intrisically independant tasks that are cirsconstancially related for a given realisation.

In an office world, contexts are especially important because many tasks last for long periods (several days or weeks are quite frequent): you begin now to prepare a business-trip that will be actually realised in two weeks, or the creation of a technical report can last for several weeks, etc. Whenever a new task has to be be planned in such a world, it is necessary to verify that there is no potential conflict with all the others running tasks. And it is desirable to take advantage of all the opportunities raised by the realisation of the other tasks. In a complex multi-agent office world, a blind verification would be completly inefficient and unrealistic: simultaneous independant tasks are numerous, and in most cases there is no need to look for interdependancy between them. Contexts are defined precisely to group together tasks that are "likely to have something to do with each other".

When a top-level activity (an activity which is not sub-part of another one and which has sub-parts) is created, a context is automatically created for itself and all its sub-activities: a tree of activities linked by the sub-part relationship is always contained in a context. Any context can be extanded by an heuristic expressed as a set of rules:

[11] This is typical opportunistic planning [Hayes 79].

If task α is in a context $\mathcal{X}$
and Running(α)
and task β (which is not sub-part of α) is created
and (Overlaps(α,β) OR During(α,β))
and α and β has a same involved agent
then put β in the context $\mathcal{X}$

If task α is in a context $\mathcal{X}$
and Running(α)
and task β (which is not sub-part od α) is created
and (Overlaps(α,β) OR During(α,β))
and α and β occur in the same room $\mathcal{R}$ [12]
then put β in the context $\mathcal{X}$

If task α is in a context $\mathcal{X}$
and Running(α)
and task β (which is not sub-part od α) is created
and (Overlaps(α,β) OR During(α,β))
and α and β make use of the same limited resource
then put β in the context $\mathcal{X}$

Properties of a *Context* are

- *Begining-date* which is equal to the begining-date of the first activity in the context;
- *Ending-date* which is equal to the ending-date of the last activity in the context;
- *List-of-agents* which is the list of all agents involved in all the activities in the context;
- *List-of-tools* which is the list of all the limited resources (tools and budget) used in all the activities in the context;
- *List-of-loc* the list of physical locations of the activities.

Each time a new activity is added to a context, these properties will be updated. Of course the interest of contexts is to focus the search for conflicts between tasks on small subsets of the whole office activity. Whenever a value is assigned to an activity property, verification of compatibility with other running activities is limited to those in the same context.

[12] *Room* is a leaf in the prototype tree of the *Localization* concept. $\mathcal{R}$ is an instance of *Room*.

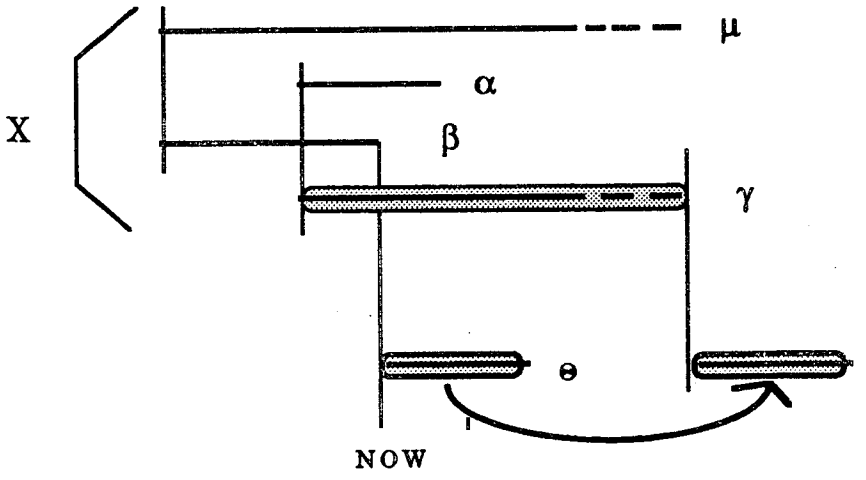

Figure 4: Context-driven planning. Case 1: γ is running.

As we have said in our introduction, the fact that we want to be able to plan tasks when monitoring running activities, implies that we do not allow to our planner to modify the past: it is impossible to withdraw any decision about ended activities, and in most cases it is impossible (or at least very costly) to modify running ones. Let's describe the main features of the algorithm directing planning activities in our world.

Planning in monitoring mode Figures 4, 5 and 6 shows various situations in which a new task θ must be planned in a world in which context $\mathcal{X}$ is already running. $\mathcal{X}$ is made of task μ and of its sub-tasks $\alpha, \beta and \gamma$. Let's suppose we have the constraint that $starts(\alpha, \gamma)$. Let's suppose there is a conflict between θ and γ: these two tasks make use of the same single-user tool. When the conflict is detected, replanning is going to be limited to the context $\mathcal{X}$. The planning decision will differ according to tasks priorities, and to the offset between their begining dates and the "*Now*" date. First of all, in any case the planner will try to solve the conflict by assigning a different resource (tool, agent, location) to task θ. This will be possible only if an adequate alternative choice is available. In this case the conflict is solved. If it is not possible, the planner looks if the conflicting task γ is already running. If it is not yet the case, it will try to modify the conflicting resource for this task γ. If the conflict cannot be solved by a change of

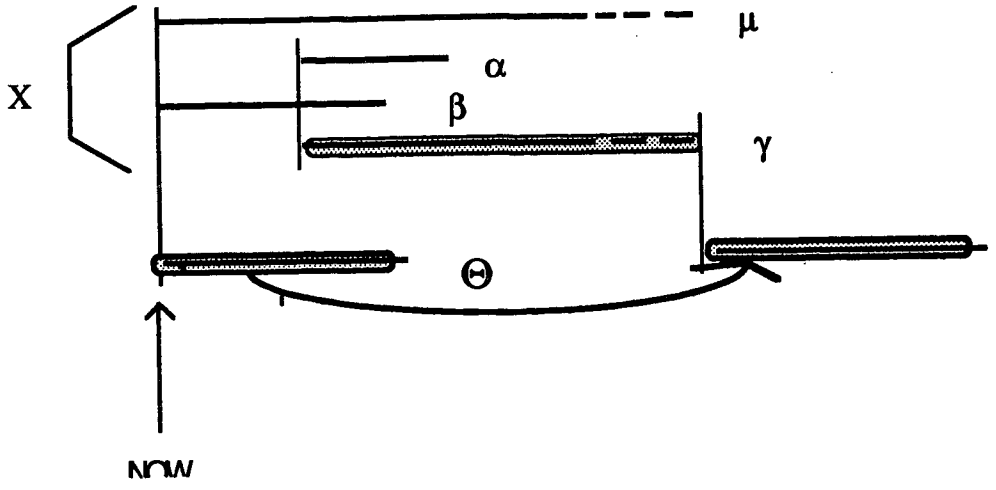

Figure 5: Context-driven planning. Case 2: γ has highest priority than θ.

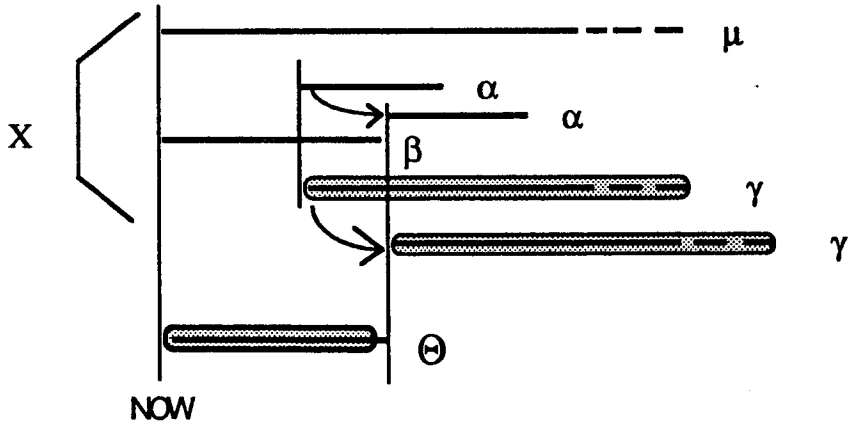

Figure 6: Context-driven planning. Case 3: γ has a lowest priority than θ.

resource in either tasks, then the planner will try to change time parameters for these tasks. The choice will be different according to the three possible situations described in Figures 4, 5 and 6.

In case no 1, γ is running and θ has a high priority: it must begin as soon as possible. The choice here will be to begin θ at the end of γ. In cases no 2 and no 3, γ is not yet running. If θ has the highest priority (compared to those of α and γ), it will begin *now*, and beginning dates of α and γ will be differed until the end of θ. If θ has the lowest priority it will begin only after the completion of γ. If priorities are equal, the shortest task (*average-duration*) will begin first.

Of course, θ is now member of context $\mathcal{X}$ and if latter a new activity has to be planned, there will be a search for potential conflict with this context.

Creating and solving new conflicts. A conflict solution can produce in some "bad" cases a new conflict. Such a case is described with its solution in Figure 7. The planner has made the choice to differ the begining of task θ after the end of the tasks belonging to context $\mathcal{X}$. As a result, context $\mathcal{X}$ has now a longuest duration and a conflict with context $\mathcal{Y}$ can appear. The solution here is to differ the begining of the first task of context $\mathcal{Y}$ after the end of context $\mathcal{X}$. This is possible because context $\mathcal{Y}$ is not currently active (no task running). Sumultaneous activity for contexts $\mathcal{X}$ and $\mathcal{Y}$ is impossible: as they share common resources, their tasks would have been merged in the same context when created.

Wait-states and external events In many occasions, the completion or creation of an activity is triggered by user-generated events: for instance, when preparing a Business-trip, the signature of a document complete an "Authorising-bt" activity, and allows the begining of hotel and travel reservation. This means first that there must be a rule in our office representation expressing that "Hotel-reservation" cannot be created before the completion of the "Authorising-bt" belonging to the same context, and second that a change in the state of a slot of a document can trigger a modification in the state of the Authorising activity and subsequently the creation of the next activities.

In the current implementation of ASTRE, these triggering rules exist, (they are constraints attached to the triggered task), but the actual firing of these rules are still "hand-made": when the planner stops (no more activities can be created until a change in the state of the world), the world-state

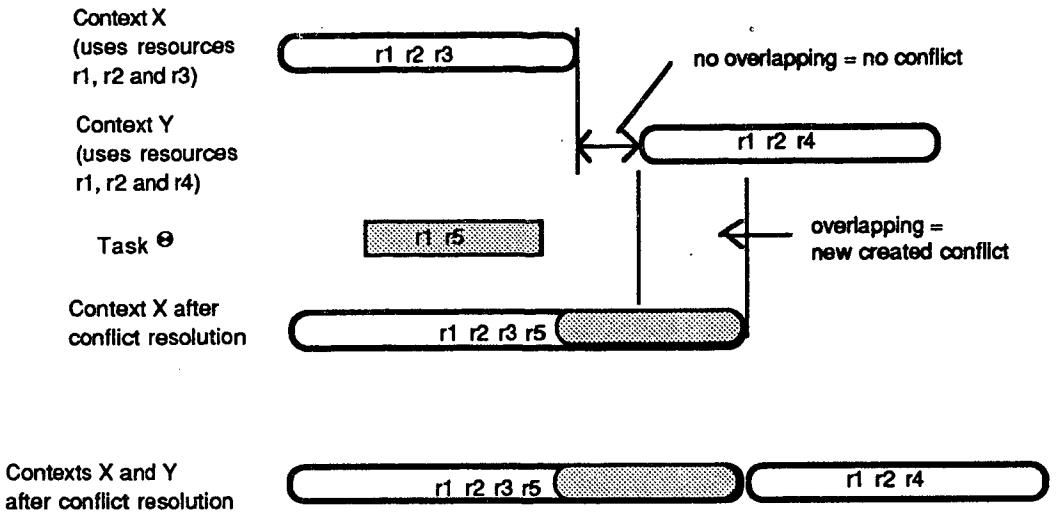

Figure 7: Side-effect conflict and its solution

is saved. The user can modify directly a value of a slot of any object in this world-state and restart the planner on this modified initial state. [13] The integration of a planner in a software office environement would necessitate a direct communication link between the office applications and the knowledge-base: creation or modification of a document should automatically modify the corresponding object in the knowledge-base. We have not adressed these integration problems yet.

4 CONCLUSION

The implementation of a planning and monitoring system is currently underway (September 88) within the SMECI [Haren 85] environment. In fact the main work that must still be done is to create an acceptable user interface enabling end-user to tailor the generic office representation to his own needs and to enter requests to the planner. We wish to propose an interface

[13] This possibility to save a world-state and stop the planner from the "action" part of a rule, to edit this state and to restart the expert system on this edited state, is offered for free by the SMECI expert-system shell.

in which visualisation of concepts and of relations among concepts will be possible through graphic support.

The resulting prototype will not be considered as an end-product but as a first step in a design-and-evaluation iterative process. Our intention is to propose to various office personnel (without any expertise in A.I.) to browse through the existing knowledge-base, to compare it with their own cognitive representation of their office environment and to try to modify it. A human factors specialist will observe users behaviour during these experimental tasks, trying to infer possible improvment in our design.

Another research and development effort will have to be done to provide useful explanations about the planning process. As we have said in the introduction, such a planner can be seen as an advice-giving system about office procedures, only if it is able to offer adequate descriptions of the plan rationale. The first step in this direction will be to define what "deep knowledge" [Clancey 83] it is necessary (and possible) to add to our knowledge-base to provide a rational basis to office procedures.

4.1 ACKNOWLEDGMENTS

We wish to thank Carolyn Foss, Martin Ader, Michel Tueni and Alain Giboin for their numerous constructive comments on a first version of this paper.

References

[Brachi 83] BRACHI G., PERNICI B.; *SOS: a Conceptual Model for Office Information Systems.* Proc of ACM SIGMOD Database Week, San Jose, May 1983.

[Clancey 83] CLANCEY W.J.; *The Epistemology of a Rule-Based Expert System - a Framework for Explanation.* Artificial Intelligence, 1983, 20, 215-251.

[Haren 85] HAREN P., NEVEU B., GIACOMETTI J.P., MONTALBAN M., CORBY O.; *SMECI: Cooperating Expert Systems for Civil Engineering Design.* SIGART Newsletter, April 1985, 92, 67-69.

[Hayes 79] HAYES-ROTH B., HAYES-ROTH F.; *A cognitive Model of Planning.* Cognitive Science, 3, 1979, 275-310.

[Kaye 87] KAYE A.R., KARAM G.M.; *Cooperating Knowledge-Based Assistants for the Office.* ACM Trans. on Office Information Systems, 5, 4, 1987, 297-326

[Lansky 86] LANSKY A.L.; A Representation of Parallel Activity Based on Events, Structure, and Causality. in *Reasoning about Actions and Plans*, M.P. Georgeff & A.M. Lansky Eds, Morgan Kaufman Pub., Los Altos, Cal., 1986.

[Rice 88] RICE R.E.; *Collection and Analysis of Data from Communication Systems Networks.* Conf. on Office Information Systems, Palo Alto, March 1988.

[Sathi 85] SATHI A., FOX M., GREENBERG M; *Representation of Activity Knowledge for Project Management.* IEEE Trans. on Pattern Analysis and Machine Intelligence, 7, 5, 1985, 531-552.

[Sebillotte 88] SEBILLOTTE S.; *Hierarchical Planning as method for task analysis: the example of office task analysis.* Behaviour and Information Technology, 7, 3, 1988, 275-294.

[Tsichritzis 87] TSICHRITZIS D., FIUME E., GIBBS S., NIERSTRASZ O.; *KNOs: KNowledge Acquisition, Dissemination, and Manipulation Objects.* ACM Trans. on Office Information Systems, 5, 1, 1987, 96-112.

[Tueni 87] TUENI M., LI J., FARES P.; *AMS: A Knowledge-based Approach to Tasks Representation, Organization and Coordination.* Conf. on Office Information Systems, Palo Alto, March 1988.

[Woo 86] WOO C.C., LOCHOVSKY F.H.; *Supporting Distributed Office Problem Solving in Organisations.* ACM Trans. on Office Information Systems, 4, 3, 1986, 185-204

[Zloof 82] ZLOOF M.M.; *Office-by-example: A business language that unifies data and word processing and electronic mail.* IBM Syst. J., 21, 3, 1982, 272-304.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique