26

Coordinates: a framework for enterprise modeling

G.S. Mannarino, G.P. Henning+ and H.P. Leone*.*
**INGAR, Fundación ARCIEN - CONICET,*
GIPSI-FRSF-Universidad Tecnológica Nacional
Avellaneda 3657, 3000 Santa Fe, Tel.: 54 42 553439,
Fax: 54 42 553439, E-mail: hleone@alpha.arcride.edu.ar
+ Instituto de Desarrollo Tecnológico para la Industria
Química (INTEC), CONICET - UNL
Güemes 3450, 3000 Santa Fe, Argentina, Tel.: 54 42 559175,
Fax: 54 42 550944, E-mail: ghenning@intec.unl.edu.ar

Abstract

This work describes advances in relation to previous contributions (Mannarino et al., 1997a, b) regarding the development of a framework for representing an organization through its different dimensions. The paper focuses on the development of *Task* and *Dynamic models. Task models* describe an organization from a functional point of view. They characterize the organization in terms of the *Tasks* that are carried out to achieve the enterprise goals and the way these activities are linked through physical and information resources and temporal relationships. *Dynamic models* depict the dynamic behavior of a production environment. They put emphasis on how the *Resources* of an organization evolve and interact to achieve complex goals.

Keywords

Enterprise Integration. Enterprise Modeling. Object Oriented Analysis.

J. J. Mills et al. (eds.), *Information Infrastructure Systems for Manufacturing II*

1 INTRODUCTION

Many strategies, such as quality improvement, business process reengineering and enterprise integration are currently employed by production organizations to cope with a highly competitive environment. Though conceptually different, they share one common aspect: the need to understand and describe the target organization through its objectives, processes, resources, costs, etc. This knowledge can be captured by developing different models of the organization. Models can be employed to describe the organization "as-is" to evaluate its processes as well as to define a new or desired organization. This work describes advances in relation to previous contributions (Mannarino et al., 1997a, b) regarding the development of a language for representing an organization through its different dimensions. *Task, Domain and Dynamic metamodels* are presented as a language for enterprise description in terms of a vocabulary, with an associated meaning, that is combined according to specific syntactic rules.

Mannarino et al. (1997a, b) focused on *Domain and Task models. Domain models* are used to identify the relevant entities of a production enterprise, to characterize them and to represent the static relationships among them. *Task models* describe the organization in terms of the *Tasks* that are carried out to achieve the enterprise goals and the way these activities are linked through physical and information resources and temporal relationships. In a *Task model*, resources may play different roles according to the behavior the task encapsulates.

This contribution extends the *Task model* previously proposed (Mannarino et al., 1997 a, b) in order to represent the fact that a *Task* may have various endings and may be carried out in different ways. Moreover, this paper focuses on *dynamic models* that describe the dynamic behavior (Ziegler, 1984) of a production environment. One of the proposed models is *the State Transition Diagram* (*STD*) that describes how a given *Resource* entity evolves during its life-cycle. A *STD* combines the attainable states of an object with the *Tasks* that can make this object change its state. Therefore, it specifies not only the attainable states of an object but also a partial order among them. In this work, the semantics of *STDs* is described and linked to *Task models*.

2 COORDINATES

2.1 Task model

Task models are proposed as a tool for describing an organization from a functional point of view (Gruninger and Fox, 1994; Schreiber et al., 1993). In order to represent an activity that has a certain characteristic duration, the *Task* construct is employed. A *Task* makes use of different resources for accomplishing a set of goals. Resources represent both the physical and information entities of the organization. Therefore, a *Task* references the resources that are *used*, *employed*, *created*, *deleted*, *consumed* or *produced* during its execution. A *Task* may depend on other tasks for its execution because of the temporal order defined among tasks. To represent this temporal order the temporal links proposed by Allen (1983) are

employed: *before*, *after*, *during*, *meets*, *overlaps*, *starts*, *finishes*, *equal* and their converses. Moreover, a given task may depend on other tasks because these last tasks supply the resources the task is requiring to be carried out. Thus, the execution of a given *Task* is constrained by both the availability of its associated resources and by temporal links.

Mode

Depending on what happens during its execution, a *Task* can have different endings: (i) it can be carried out as planned, (ii) it may partially meet its goals because something goes wrong during its execution, (iii) it may be finished prematurely, etc. For instance, the *Task* of *Controlling the quality of a batch of product* can result in two situations, either approving or rejecting the particular batch. To represent the fact that a *Task* can have several endings, the **Mode** class is defined. Figure 1 shows some possible types of endings. A *Task* is associated with a given *Mode* through the *task-mode-link* (Figure 2). The zero or more cardinality at the end of the *Mode* construct indicates that a *Task* may have different endings. However, a *Mode* is associated to a unique *Task*. This reflects the fact that, even when the two different tasks have the same conceptual ending, e.g., *Meets its goals*, the results of their execution are distinct. Moreover, as a task actually assumes only one final status, its associated *Modes* are linked through an "or-exclusive" structure.

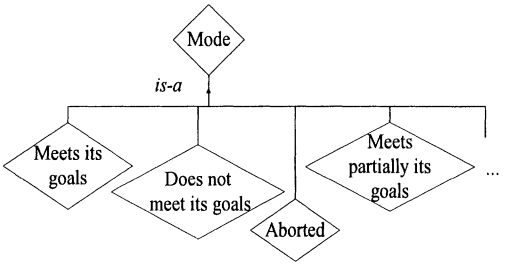

Figure 1 Different types of modes.

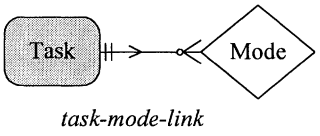

Figure 2 A Task can assume different endings.

Resource

Tasks are chained by a set of physical and information *Resources*. A *Resource* can be a machine, an employee, a document, a product, etc. (see Mannarino et al., 1997). A *Resource* may assume different roles, depending on the tasks in which it participates. To this end, the *ResourcePerspective* (Figure 3) entity is defined. It represents a view of a given *Resource* and filters only those *Resource* characteristics that are of interest in a given context. Consider, for instance, an equipment item: the information that is relevant to the Engineering Department (functional and material characteristics) is different from the one that is important to the Maintenance Department (preventive plans, history of faults and corrective actions, shutdown periods, etc.). Similarly, the information that is pertinent to the Purchasing Department (alternative suppliers, technical characteristics, costs, etc.) is distinct from the one the Production Planning and Control Department is interested in (*Tasks* it can execute, products that can be processed, etc.).

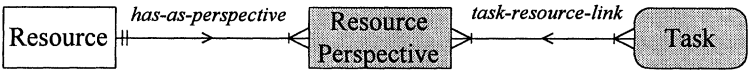

Figure 3 A *ResourcePerspective* filters certain characteristics of a *Resource*.

As shown in Figure 3, a *Task* references the *Resource Perspectives* that participate during its execution. The effect of a given *Task* over a *Resource* is actually specified by the *task-resource-link* (Figure 4) which generalizes the *uses, consumes, produces, eliminates* and *creates* links. A *Resource* is *consumed* if it is transformed (according to a conservation law) into one or more *Resources*. Conversely, a *Resource* is *produced* if a certain amount of it appears (according to a conservation law) when the *Task* has finished. Thus, the *consumes* link requires the existence of one or more *produces* links. Similarly, a *Resource* is *used* if it remains unchanged when the *Task* has finished. The *creates* (and its inverse *eliminates*) relationship implies that a new *Resource* appears in the domain (disappears) as a consequence of the *Task* execution and it is applied to those *Resources* that do not satisfy conservation laws.

The structure of a *task-resource-link* encapsulates the characteristics of a *ResourcePerspective* that are relevant to a given *Task*. But how are these characteristics identified? By resorting to the concept of **state**. An object is defined by a set of slots or attributes, which take on some values. Some of these attributes may change their values over time as a consequence of receiving different stimuli. Thus, the *state* of a *Resource* is defined by a set of <attribute-value> pairs at a specific point in time. A *Resource*, and consequently a *ResourcePerspective* is associated to its possible states through the *resource-state* link. For a given *Task* to be performed, its associated *Resources* have to be at specific states, known as pre-states. Similarly, the *Task* execution may cause changes of the resource states, given rise to the so-called post-states. Consequently, the effects of a given *Task* are specified in terms of a set of *Resources* that may change their states. Figure 4 shows that a *Task* actually represents its effects over a given *Resource* by making reference to specific states of the *Resource*:

(i) The *initial state* relationship references the required state of the resource to participate in the task,
(ii) The *final state* relationship references the state the resource assumes when the task has finished,
(iii) The *intermediate state* encapsulates the evolution of the resource during the task execution.

Each type of relationship generalized in the *task-resource-link* (*uses*, *consumes*, etc.) imposes specific constraints over the states that are referenced by the *initial-state*, *intermediate-state* and *final-state* relationships. In particular, the *uses* link requires the initial and the final states of the *Resource* to be the same. Similarly, the *produces* link makes a *Resource* evolve from not existing at all to existing in a certain amount. Conversely, the *consumes* link makes a *Resource* evolve from existing in a given amount to exist in a smaller amount or to not existing at all. The

links *creates* and *eliminates* share the same semantics as the *consumes-produces* relationships, except that they are applicable to *Resources* that do not satisfy conservation laws.

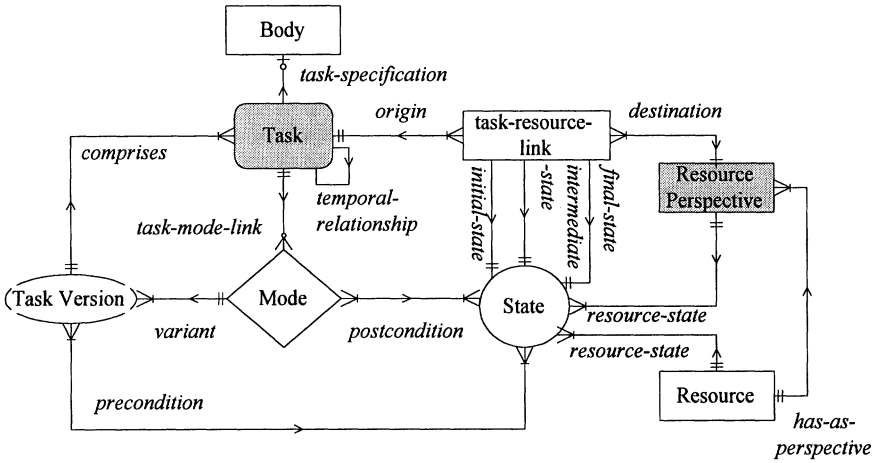

Figure 4 Task metamodel.

As it was already mentioned, a *Mode* (Figure 4) construct characterizes a specific Task ending. But how is a particular *Task* ending actually described? By making explicit the final states of the task associated *Resources*. Thus, a *Mode* is linked to a set of *Resource* states through the *postcondition* link.

Version

A *Task* can adopt several forms in a given *Mode*: it can be accomplished by resorting to alternative *Resources*, different subtasks may be executed to reach the same goal, various methods may exist to solve the same problem, etc. Therefore, the *TaskVersion* construct is included in the language to encapsulate a particular decomposition task structure for carrying out a given *Task* under a particular *Mode*. To show the fact that a *Task* can reach the same final status by resorting to different task structures, a *Mode* is linked to the *TaskVersion* construct through the *variant* relationship (Figure 4).

Tasks can be solved by the execution of one or more subtasks. Since the *TaskVersion* construct represents the aggregation concept that relates a *Task Mode with* a collection of subtasks, the *comprises* relationship links a given *TaskVersion* with the set of one or more subtasks it is decomposed into. *Tasks* can be disaggregated at any level, depending on the problem being analyzed. When the required level of decomposition is reached, a *Task* is specified in terms of a script or conventional procedure encapsulated in the *Body* class. Thus, a *Task* (for a given *TaskVersion* under a particular *Mode*) can be described in terms of either a body or a set of subtasks, but not simultaneously both. Therefore, the following invariant is associated to the *Task* class.

Invariant(Task) $\equiv (\forall T \in Task, \forall B \in Body, \forall M \in Mode)$
task-mode-link(T, M) $\rightarrow \sim$ *task-specification(T, B)*

A *TaskVersion* is constrained by the use of a given set of *Resources*, as specified in the *Task* it implements. Moreover, *TaskVersion preconditions* make reference to those *Resource* states that are needed for the execution of a *Task with* a particular structure. The following rules define the relationships among a *Task*, the subtasks *comprising* its *TaskVersion,* and the set of associated *Resources*:

- If a *Task* is an elementary one, **all** the *Resources* required for performing it have to be referenced by the *Task*. They have to be in the specified *initial states* **before** performing the *Task*. The *Body* construct is responsible for accomplishing the *Task* semantics by sending messages to the *Task* associated *Resources*.
- If a *Task* is not an elementary one, it is not necessary to specify all the *Resources* related with the *Task* at its corresponding level of abstraction, but where the resources are precisely used. The place a *Resource* is required to be described is actually determined by the subtask that first makes reference to it. As a *Mode* specifies the final states of a set of *Resources*, it not only constrains the *Resources* that take part in its subtasks but also the *Modes* associated to these ones. Similarly, as a *Resource* is not necessarily specified at a given abstraction level, the *Resource* final status can be inferred by searching for those subtasks that ultimately have an explicit relation with the *Resource*.

2.2 Dynamic models

State Transition Diagram (STD)

Task models represent *Tasks* from a static point of view; i.e., a *Task* is expressed in terms of a set of subtasks and *Resources* that collaborate to meet the *Task* goals, as encapsulated in the *Mode* construct. *Dynamic models* put emphasis on how a given *Resource* entity or a set of *Resource* entities evolve during a given period of time and on how they interact to achieve *Task* goals. As mentioned earlier, a *Task* may change the states of the *Resources* it is connected to in a given model. It was also mentioned that the link that relates a *ResourcePerspective* with a *Task* identifies the *Resource* states of interest to the *Task* at hand. *State Transition Diagrams* (*STD*) (Harel and Naamad, 1996; Harel and Gery, 1997) are employed in order to represent how a *Resource* evolves over time from state to state. Under the context of this contribution, a *STD* is defined as a digraph where *nodes* represent *Resource* states and *arcs* are labeled by messages that cause the transition of a *Resource* from a pre-state to a post-state. In turn, messages are responsible for starting and finishing *Tasks*. Thus, a *STD* specifies the possible sequences of *Tasks* that might affect a *Resource* during its life-cycle. Figure 5 shows that a *Transition* links a pair of *State* objects. A *ResourcePerspective* can change its state as a consequence of a message that starts or finishes a given *Task* under a particular *Mode.* This change of state is identified by the *trigger* link that relates the *task-resource-link* with the *Transition* construct.

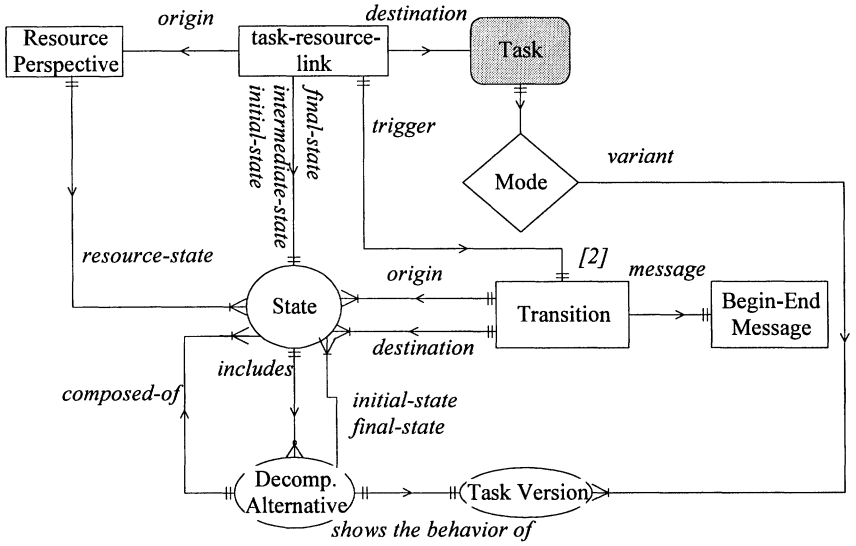

Figure 5 Dynamic metamodel.

The cardinality of the *trigger* link shown in Figure 5 indicates that each *task-resource-link* may only have two associated *transitions*, one from the initial state to the intermediate one and another from this last state to the final one. As a *TaskVersion* describes a particular way of accomplishing a *Task*, the *Resources* that participate in a given *Task* may evolve through different sequences of intermediate states, depending on the structure of the *Task*. A *Resource* state may abstract many alternative scenarios. These scenarios are *STDs* that represent the *Resource* evolution in alternative *TaskVersions*. The *Decomp.Alternative* class is used to decompose a state into a particular *STD*. Each *Decomp.Alternative* is associated with a *TaskVersion*. States can be expanded to any level of detail, according to the number of disaggregation levels of their associated *Tasks*. When a *State* is disaggregated into a set of substates the semantics of such a transition is as follows. When a *Resource* evolves from a state to a composite state, it will traverse through the states defined by the chosen *TaskVersion*. The initial-state of the *Decomp.Alternative* option will be the first state the *Resource* will enter in. Later, it will evolve through the initial state successors till reaching the final-state. Finally, after the *Resource* reaches the final-state, it will traverse to the successor of the composite state that is defined at the upper level of abstraction.

2.3 Example

The following paragraph describes the *Sales Order Processing* scenario in an organization that has a production system based on a make-to-order strategy, i.e., a production plan is created once a week based on the *Sales Orders* received during the week. The process consists of the following activities: (i) Sales Order Preparation, (ii) Special-terms approval, (iii) Order Processing, (iv) Product

Manufacturing, (v) Quality Control, (vi) Shipping and (vii) Customer billing. A brief description of two of them is given below:

(i) *Sales Order Preparation*: a *Salesperson* receives a customer call and fills out an electronic *Sales Order Form*. The form contains information such as order number, sales person-id, customer-account, date, product-id, description, quantity, and special terms. The *Sales Supervisor* may also receive customers' calls, in case he/she is available. Once the *Sales Order Form* has been completed, it is incorporated into a List of Production Orders to be Scheduled (*LPOS*) unless the order contains special-terms items, in which case the order is forwarded to the *Sales Supervisor* for approval before including it in the LPOS. This list is consulted during the week by the *Scheduler* so as to get an overall view of the production requirements and to suggest limits on future sales. The LPOS is a repository of the Sales Orders received during the week and contains for each order, customer-id, required amount, priority, desired reception due date, etc..

(ii) *Special-terms evaluation*: The *Sales Supervisor* double-checks products which have special terms due to quantity discounts, special sales, etc.. The special-terms can be authorized or rejected by the *Sales Supervisor*. When they are not approved, the *Sales Supervisor* contacts the customer to negotiate new conditions. If new conditions cannot be agreed, the sales order is canceled. Special term conditions can be accepted because of (i) the customer has a reliable payment history, (ii) the customer makes an important purchase and provides good references from other suppliers, (iii) the customer makes an important purchase and provides a guarantee, or (iv) new conditions are negotiated.

(iii) *Order Processing*: .....

(iv) *Product Manufacturing*: ....

(v) *Quality control:*.....

(vi) *Order shipping*:.....

(vii) *Order billing*.....

To show the use of the *Mode* concept, consider the situations that can occur during the Sales Order (SO) special-terms evaluation. Two alternatives can arise when evaluating a client sales requirement: (i) the customer sales conditions are accepted, and (ii) the customer sales conditions are rejected. The task *Negotiate special terms* is created to represent the *Special-terms evaluation* activity. Figure 6 shows the fact that the task *Negotiate special terms* can have two ending status': *SO: requirements accepted*, and *SO: canceled*. In the latter case, the task does not meet its goals if the customer requirements are rejected by the *Sales Supervisor* (SS) and thus, the *Sales Order* is canceled. Moreover, the person responsible for dealing with the customer is the *Sales Supervisor*, as expressed by the $uses_1$ link: he/she needs to be *available* for evaluating the customer requirements, he/she will evolve to the state *analyzing customer requir.* as soon as he/she starts analyzing those requirements, and return to the *available* state when the task has finished. What does it mean that the *Sales Supervisor* is in the *analyzing customer requir.* state? To answer this question, the state needs to be disaggregated according to the *Mode* adopted for the task *Negotiate special terms*, as it will be discussed afterwards.

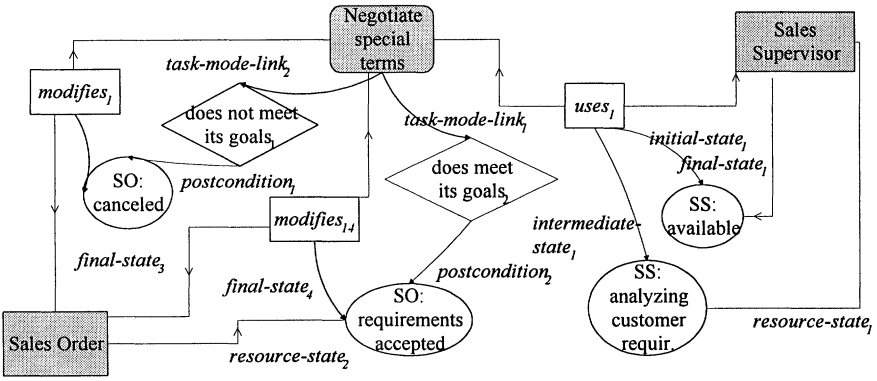

Figure 6 Special-terms requirements can be either accepted or rejected.

Figure 7 shows that, in order to prepare a *Sales Order*, the task *Negotiate special terms* has to be performed, in case the *Sales Order* has some special sales conditions. This task can result in either preparing a SO or canceling it. The Sales Order processing specifies that the SO can be filled out by either the *Sales Supervisor* or a *Sales Person.* Task version *Special terms exist and SalesSup.* encapsulates the alternative in which the *Sales Supervisor* handles the communication with a customer that requires some special sales conditions. On the contrary, the alternative *Special term exist and SalesPer.* identifies the case in which a *Sales Person* handles the communication with a customer and special sales conditions are requested.

As seen in the description of the Case study, special terms conditions can be accepted due to different reasons. To encapsulate them, the Task versions *Payment history*, *Important purchase and good references, Important purchase and guarantee provided* and *Special terms initially rejected and later approved* are created. The alternative *Special terms exist and SalesPer.* is disaggregated in Figure 7. What does characterize this alternative? The fact that (i) a *Customer* contacts a *Sales Person*, (ii) the *Customer* requires some special sales conditions, and (iii) these conditions are accepted by the *Sales Supervisor* because of the terms mentioned above. As the *Sales Person* deals with the customer, the *Sales Order* has to be forwarded to the *Sales Supervisor* for its evaluation. The *meets* relationship that links the tasks *Fill out sales order* with *Forward to sales supervisor* expresses the fact that as soon as the SO is filled out it has to be sent to the *Sales Sup.* in case it requires authorization. Furthermore, the task *Negotiate special terms* is required to accept the sales conditions no matter how one gets to it (through one of the Task Versions *Payment history*, *Important purchase and Good references, Important purchase and Guarantee provided* and *Special terms initially rejected and later approved*).

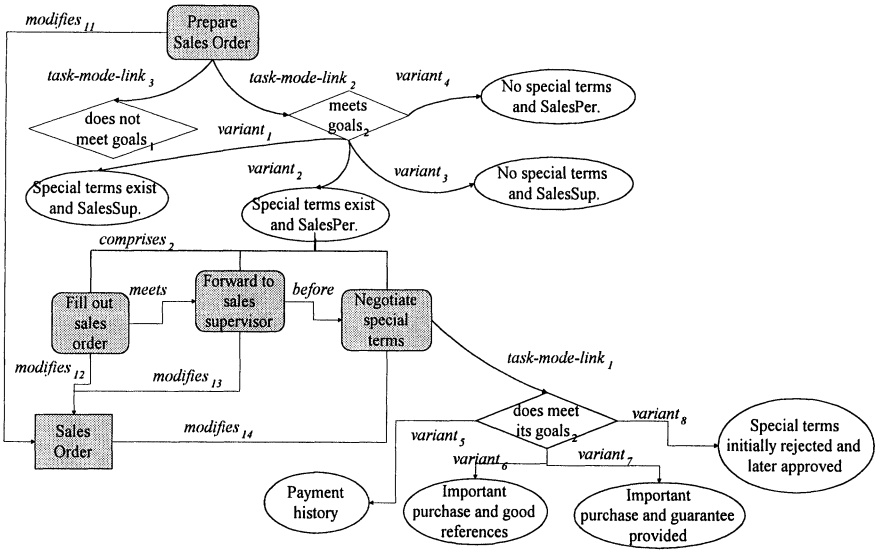

Figure 7 Different alternatives of the task *Prepare Sales Order*

In this example, the task *Prepare Sales Order* modifies a particular *Sales Order*; this is equivalent to say that the Sales Order evolves from the *unprepared* state to the *prepared* state, passing through the *in preparation* state. Figure 8 shows that being *in preparation* means evolving through the series of states "empty", "filling out", "filled out and req. authorization" ... "requirements accepted". The *in preparation* state is then a composite one that encapsulates a complex structure. The semantics associated with the composite state *SO: in preparation* is as follows: The composite state is entered when the task *Prepare SO* is started (*Begin(modifies11)*). The set of substates are then chained by the subtasks of *Prepare SO* identified in the task model, starting from the initial state *SO: empty* (the first in the sequence). When the final state is reached (the one drawn with two concentric ellipses) the *End(modifies14)* makes the SO become prepared.

3 CONCLUSIONS

Task models represent the processes of an organization in terms of a set of *Resources* that are transformed. The use of the *Mode* and *Task Version* concepts enrich the description of activities extending the aggregation relationships frequently used in modeling formalisms to represent different levels of detail. The *Task model* associates to each level of abstraction a set of goals (or final status) and preconditions that characterize the task and put constraints on its internal structure. These extensions to the Coordinates language are represented in the *Task model* through both the *Mode* construct, by considering all the relevant final endings associated to a *Task,* and the *TaskVersion* entity, that makes explicit all the alternatives of getting into these different endings. To effectively manage the *Resources* of an organization, it is necessary to identify its *Resources* and their

behaviors. The *Task* model has focused on the resource management by integrating the *Resource STDs* to the *Tasks'* descriptions. A *ResourceSTD* is the dynamic facet of models that puts emphasis on how a particular resource evolves during its life-cycle. *Complex resource states* are used to encapsulate multiple alternative scenarios found in a given *Resource* life cycle. This aggregation relationship is represented in the *Task model* by the *Decomp.Alternative* entity that links a *Resource* state with a *STD*, under a particular Task Version.

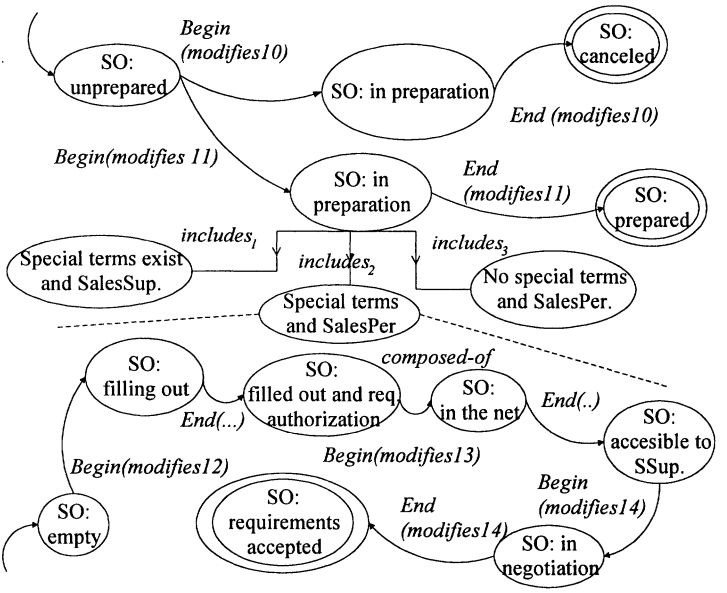

Figure 8 Evolution of the resource *SO* when it participates in the *task Prepare Sales Order.*

4 REFERENCES

Allen J.F., (1983), Maintaining knowledge about temporal intervals. *Communications of the ACM*, **26**, 832-843.

Gruninger, M. and Fox, M.S. (1994) An activity Ontology for Enterprise Modelling. Workshop on Enabling Technologies – Infrastructures for Collaborative Enterprises, West Virginia University.

Harel, D. and Naamad A. (1996) The STATEMATE Semantics of Statecharts. *ACM Trans. Soft. Eng. Method.*, 5:4.

Harel D. and Gery, E. (1997) Executable Object Modeling with Statecharts. *IEEE Computer*, July 1997, 31-42.

Mannarino, G.M.; Henning, G.P. and Leone, H.P. (1997a) Process Industry Information Systems: Modeling Support Tools. *Computers and Chemical Engng.*, **21**, Suppl., S667-S672.

Mannarino, G.M.; Henning, G.P. and Leone, H.P. (1997b) Metamodels for information system modeling in production environments, in *Information Infrastructure Systems for Manufacturing*. (ed. J.B.M. Goosenaerts, F. Kimura y H. Wortman), IFIP- Chapman & Hall.

Schreiber, G.; Wielinga, B. and Breuker, J., (1993) KADS A Principled Approach to Knowledge-Based System Development. Academic Press Inc.

Ziegler B.P., (1984) *Multifacetted Modelling and Discrete Event Simulation.* Academic Press, London.

5 BIOGRAPHY

Gabriela S. Mannarino is a Research Fellow of "Consejo Nacional de Investigaciones Científicas y Técnicas" (CONICET) at INGAR (CONICET - Fundación ARCIEN). She holds an Information Systems Engineering Degree from "Universidad Tecnológica Nacional, Fac. Reg. Santa Fe" (1993). She is currently working on object oriented analysis and design methodologies and the development of integrated information systems for production environments.

Gabriela P. Henning is an Associate Professor and Researcher at INTEC (CONICET-Universidad Nacional del Litoral), Santa Fe, Argentina. She received a Chemical Engineering Degree from "Universidad Tecnológica Nacional, Fac. Reg. Rosario" (Argentina) in 1981 and earned her Ph.D. in Chemical Engineering from "Universidad Nacional del Litoral" in 1986. She was a Postdoctoral Fellow at the Massachusetts Institute of Technology (1986-1989), specializing in Applications of Artificial Intelligence in Process Engineering. Her research interests include the application of knowledge-based and mathematical programming methodologies to industrial scheduling problems, the development of high-level languages for process engineering activities and the design of integrated information systems for production environments.

Horacio P. Leone holds positions as a Full Professor at the Information Systems Department, Universidad Tecnológica Nacional Fac. Reg. Santa Fe, and Researcher at INGAR (CONICET-Fundación ARCIEN), Santa Fe, Argentina. He received a Chemical Engineering Degree from "Universidad Tecnológica Nacional, Fac. Reg. Rosario" (Argentina) in 1981, and earned his Ph.D. in Chemical Engineering from "Universidad Nacional del Litoral" in 1986. He was a Postdoctoral Fellow at the Massachusetts Institute of Technology (1986-1989), specializing in Applications of Artificial Intelligence in Process Engineering. His current research focuses on object-oriented analysis and design of information systems for process industries.