

# A notation for malleable and interoperable coordination mechanisms for CSCW systems

Carla Simone, Monica Divitini

Dept. of Computer Sciences  
University of Milano  
Via Comelico, 39/41  
I-20135 Milano, Italy  
{simone, divitini}@hermes.mc.dsi.unimi.it

Kjeld Schmidt

Cognitive Systems Group  
Risø National Laboratory  
P. O. Box 49  
DK-4000 Roskilde, Denmark  
kschmidt@risoe.dk

**ABSTRACT.** In order to handle the high degree complexity that characterizes cooperative work in modern work settings, the articulation of the distributed activities requires a category of symbolic artifacts that stipulate and mediate articulation work. We call these artifacts ‘coordination mechanisms’. From the evidence of sociological field studies, it is evident that computational coordination mechanisms must be malleable and interoperable. The paper describes a notation for constructing computational coordination mechanisms that meet these requirements.

## 1. INTRODUCTION

In Computer-Supported Cooperative Work (CSCW), a crucial issue is to devise computational coordination facilities that support cooperating actors in managing the complexity of articulating their distributed and yet interdependent activities.

Cooperative work is constituted by multiple actors who are interdependent in their work and who therefore have to divide, allocate, coordinate, schedule, mesh, interrelate, integrate, etc. — in short: *articulate* their individual activities: Who is doing what, where, when, how, by means of what, under which constraints? [17; 35]. Because multiple actors are involved, cooperative work is inexorably and fundamentally distributed, not only in the usual sense that activities are distributed in time and space, but also — and more importantly — in the sense that actors are semi-autonomous in terms of strategies, heuristics, perspectives, conceptualizations, goals, motives, etc. [30].

With low degrees of complexity, the articulation of cooperative work can be achieved by means of the modes of interaction of everyday social life. In fact, under such conditions, the required articulation of individual activities is managed *so* effectively and effici-

ently by our repertoire of intuitive interactional modalities that the distributed nature of cooperative work is not apparent — most of the time. However, with the complex work environments that characterize modern industrial and administrative organizations, the task of articulating the complexly interdependent and yet distributed activities is of a different order of complexity.

In order to handle a high degree complexity of articulation work, the articulation of the distributed activities of cooperative work requires support by means of a category of symbolic artifacts which, in the context of a set of procedures and conventions, stipulate and mediate articulation work and thereby are instrumental in reducing the complexity of articulation work. We call these artifacts and the concomitant procedures and conventions ‘coordination mechanisms’.

The concept of coordination mechanisms has been developed as a generalization of phenomena described in different ways in different empirical investigations of the use of artifactually embodied protocols for the articulation of cooperative activities in different work domains, e.g., standard operating procedures in administrative work [36; 38]; classification schemes for large repositories [1; 4]; checklists [9]; time tables in urban transport [19]; flight progress strips in air traffic control [18]; production control systems in manufacturing [31]; planning tools for manufacturing design [7]; and fault correction procedures in engineering and software design [6].

A coordination mechanism (CM) can be defined as a protocol, encompassing a set of explicit conventions and prescribed procedures and supported by a symbolic artifact with a standardized format, that stipulates and mediates the articulation of distributed activities so as to reduce the complexity of articulating distributed activities of large cooperative ensembles.

In other words, and less condensed, to serve the purpose of reducing the complexity of articulation work, a CM must have the following characteristics:

- (1) A CM is essentially a *protocol* in the sense that it is a set of explicit procedures and conventions that stipulate the articulation of the distributed activities.
- (2) The stipulations of the protocol are, in part at least, conveyed by the *symbolic artifact* and they are thus persistent in the sense that they are, in principle, accessible independently of the particular moment or of the particular actor.

- (3) At the same time, the symbolic artifact *mediates* the articulation of the distributed activities in the sense that any change to the state of execution of the protocol is conveyed to other actors in some form by means of changes to the state of the artifact.
- (4) The symbolic artifact has a *standardized format* that reflects pertinent features of the protocol and thus provides affordances to and impose constraints on articulation work.
- (5) The state of the artifact is *de-coupled* from the state of the field of work in the sense that changes to the state of the field of work are not automatically reflected in changes to the state of the execution of the protocol and, conversely, that changes to the state of the execution of the protocol are not automatically reflected in changes to the state of the field of work.

Similarly, a *computational coordination mechanism* (C-CM) can be defined as a computer artifact that incorporates aspects of the protocol of a coordination mechanism so that changes to the state of the mechanism induced by one actor can be automatically conveyed by the artifact to other actors in an appropriate form as stipulated by the protocol.

From the evidence of the corpus of empirical studies of uses of artifactually embodied protocols for articulating cooperative activities, we have derived a set of general requirements for C-CMs and, by implication, for a general notation for constructing such C-CMs [32].

Since coordination mechanisms, as plans in general, are “resources for situated action” [37], a CM must be *malleable* in the sense that it supports users in specifying its behavior. In other words, a C-CM must provide facilities for actors to specify and respecify the behavior of the mechanism so as enable actors to meet changing organizational requirements as well as to control the execution of the mechanism by making local and temporary changes to its behavior, for instance by suspending or overruling a step, by ‘rewinding’ a procedure, escaping from a situation, or even restarting the mechanism from another point.

In order for actors to exercise control of the execution of the mechanism and respecify its behavior, the specification of the behavior of the mechanism must be accessible and manipulable to actors and, more specifically, accessible and manipulable *at the semantic level of articulation work*. That is, the objects and functional primitives offered by the mechanism must be expressed in terms of operations of articulation work with respect to roles, actors, tasks, activities, conceptual structures, resources, and so on (cf. the model of articulation work in Figure 1).

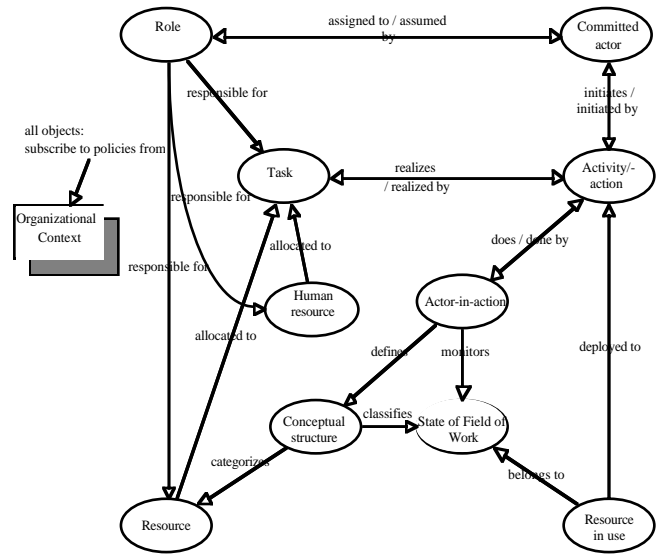


Figure 1. Objects and elemental operations of articulation work. ‘Missing links’ between objects can be constructed indirectly, by creating composite operations.

Coordination mechanisms are ‘local and temporary closures’ [17]. No single mechanism will apply to all aspects of articulation work in all domains of work. Accordingly, a C-CM should be conceived of as a specialized software device embedded in an application so as to support the articulation of the distributed activities of multiple actors with respect to the field of work as represented by that application. As an embedded system, a C-CM must thus provide means of identifying pertinent features of the field of work as represented by the data structures and the functionality of the application in which it is embedded.

Now, articulation work is a recursive function in the sense that articulation work must itself be articulated, and so forth [17]. Articulation work is done by the cooperating actors as part and parcel of their cooperative effort. Thus, it should be possible to modify the behavior of a C-CM while it is running, without having to suspend all activities within the cooperative ensemble for some time. A C-CM must thus provide means for dynamic reconfiguration of the protocol and must give actors means of controlling the propagation of changes to the behavior of the mechanism. Furthermore, since articulation work is done by the cooperating actors as part and parcel of their cooperative effort, the secondary articulation activities (of articulating articulation work) may themselves be conducted in a cooperative fashion. Thus, modifying the behavior a CM may be a cooperative activity.

This *recursiveness of articulation work* has radical implications: In order to handle the complexity of articulating the activities involved in modifying a particular CM (e.g., controlling the propagation of changes) it may be necessary to invoke another CM. Thus, it should be possible for one C-CM may take another C-CM as its ‘field of work’.

Finally, since coordination mechanisms are conceived of as ‘local and temporary closures’ and no single mechanism will apply to all aspects of articulation work in all domains of work, a C-CM *must be able to interoperate* with other C-CMs in a wider organizational field. A system of C-CMs can therefore be conceived of as a highly distributed system of software devices that are embedded in domain-specific applications and that, at the same time, may need to interoperate. Recent field studies [33] indicate that the following categories of interoperation must be supported:

- for a particular C-CM, ‘foreign’ C-CMs may provide indexing facilities for accessing resources in the wider organizational field;
- a particular C-CM may subscribe to policies and other definitions issued by other C-CMs;
- a particular C-CM may trigger other C-CMs into action;
- one C-CM may provide a control facility for cooperatively managing changes to another C-CM.

It is obvious that malleability and interoperability are contradictory requirements, since the first is based on the possibility of adapting behavior and the second one on the stability of the behavior of other mechanisms. This conflict is unavoidable but can be alleviated in different ways, however: On the one hand through negotiations among the cooperative ensembles involved, perhaps governed by suitable coordination mechanisms, and on the other hand by making coordination mechanisms tolerant to (limited) modifications.

### 1.1. Related research

The aim of providing ‘structured’ support for the articulation of distributed activities is shared by many researchers within CSCW, of course. However, most of the coordination mechanisms incorporated in CSCW systems so far are experienced as excessively rigid, either because the embedded CM is not accessible to the actors and cannot be changed, e.g., The Coordinator [15; 39], or because the facilities for changing the mechanism do not support respecification of the mechanism by the actors themselves, in their own terms, e.g., DOMINO [22; 23]. Now, a number of recent research projects attempt to make CSCW applications flexible at the semantic level of articulation work, e.g., EGRET [20] and ConversationBuilder [21]. The ConversationBuilder, for example, was developed as a “generic framework for open, active, and flexible support for collaboration”. This flexibility is achieved by providing appropriate ‘mechanisms’ for the support of collaboration rather than specific policies: “Policies can be built out of mechanisms, if the right mechanisms are provided”[3].

While closely related, our approach differs in one important respect, namely in the attempt to develop a general notation by means of which any C-CM can be specified and which, at the same time, supports the specification of coordination mechanisms in terms of articulation work, by the actors themselves and in a cooperative manner.

A different approach was taken by Malone et al. with OVAL [26]. Contrary the ConversationBuilder and EGRET, the OVAL specification language is intended to be a general notation for designing CSCW systems by providing a set of basic primitives such as objects, views, agents, and links. The OVAL notation is problematic, however, in that the primitives provided by the notation are not at the semantic level of articulation work. By contrast, the aim of the present approach is to support actors in specifying and respecifying the CM in terms of their everyday cooperative activities.

## 2. THE RATIONALE BEHIND THE NOTATION

The two main requirements stated in the introduction, namely the requirements of malleability and interoperability, drove the development of the notation for the design of C-CMs.

First of all, the requirement of malleability can be fulfilled only if modifications of the protocol modeling the C-CM can be made: at the level of the representation language, to get an expressive power appropriate to the target protocol (the grammar); at the level of the protocol itself, to redefine its structure and behavior

(permanent changes); and finally, at the level of the protocol instance, to adapt it to contingent situations (local control).

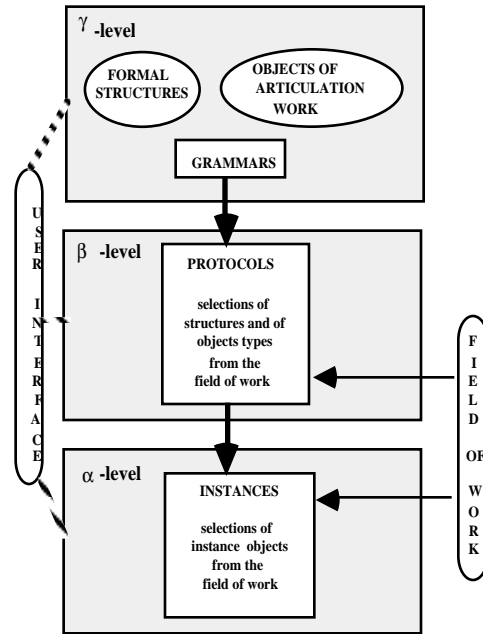


Figure 2. The layered notation and its context.

This leads to conceive of a notation for C-CMs layered in three levels, as depicted in Figure 2. Each level defines the ‘space of possibility’ of the lower level. At the  $\alpha$ -level, one can instantiate the protocols provided by the  $\beta$ -level, whereas the  $\gamma$ -level provides the grammars that can be applied at the  $\beta$ -level to define such protocols. The ‘space of possibility’ within which grammars can be defined at the  $\gamma$ -level is determined by the set of *objects of articulation work* (OAW) as defined by the model of articulation work, and by the set of *formal structures* to represent various relations (causal, part/whole, etc.) among the objects. The selection of these two types of basic elements is motivated by the need to make visible and available to the designer a set of linguistic building blocks at the semantic level of articulation work together with a formally defined way of composing them into a protocol. Moreover, since protocols model the behavior of a C-CM, namely its associated procedures and conventions, the notation is able to represent control flow through suitable and *formally defined control structures*. Formality is a key issue in order to manage modifications in a sound way since only a formal semantics can support consistency checks and provide for the evaluation of the impacts of the modifications. Finally, the  $\alpha$  and  $\beta$ -level of the notation provide a set of *primitives* for building C-CMs.

Since the notation is composed of a selection of basic elements (the OAW, their relations and the formal structures to describe the dynamic aspects of the protocols), its richness and usability depend on the flexibility by which these various components can be composed to describe and to combine different protocols. Moreover, the modifications can be done more easily and less expensively if the components of the notation can be isolated and substituted, thus reducing the impacts the changes may have on other components. To this aim the notation shows a *great degree of modularity* at the representation level as well as at the level of the operational semantics.

Modularity is also a basic means to get interoperability among C-CMs. More specifically, interoperability of C-CMs is constructed incrementally from the capability of basic elements to ex-

change information. To this aim the notations contains at the  $\gamma$ -level an Interoperability Language [11] whose components reflect the various modes in which C-CMs can be related. The *reference* mode allows a CM to contribute to the definition of another CM typically, through the OAWs that this latter is constituted of or through the subscription of policies encompassed by another CM. The *awareness* mode expresses the notification of relevant events among CMs. The *accommodation* mode expresses the mandatory exchange of information among CM's. The *over-rule* mode expresses the change of the protocols by other CMs: this allows for the representation of cooperative modification. The *control* mode allows for the definition of relations between the behaviors of compound mechanisms (e.g., synchronization).

In addition, a C-CM does not live in a vacuum: indeed, as a software device it is embedded in a context constituted by, most importantly, the Field of Work (FoW). At the  $\gamma$ -level there are no connections with the FoW. In fact, building a grammar means selecting a language to define a class of C-CMs: more specifically, which components constitute the C-CM together with their structural interrelation (e.g., in order to model workflows, classification schemes, etc.), and the possible alternative behavior of the C-CM through the definition of operational semantics associated (compositionally) to the components of the grammar. The grammars are independent of the FoW in that they simply define the expressive power of the languages potentially used to define of specific protocols.

By contrast, the definition and use of protocols are related to the FoW in many ways. In fact, the objects of the FoW are related to the objects of the articulation work both as 'types' and as 'instances' at the  $\beta$  and  $\alpha$  levels, respectively. Within protocols, "types" are imported from the FoW together with the related "methods" that are conveyed to the "instances" in the standard way.

On the other hand, the notation is fully disjoint from any notation used in the design of User Interface. This means that the proposed notation has not to be assessed from the user point of view. To the contrary, it has to be evaluated from its capability of supporting malleable C-CMs and their interoperability. In this view, the connections with the User Interface shown in Figure 2 have to be interpreted as requirements that the notation 'imposes' on the UI design: namely, that the objects populating the UI must show the same degree of malleability and modularity as the notation does.

The development of the notation concentrated primarily on the formal specification of all its components and on its check against the demands raising from the many considered field studies. This was conceived of as a fundamental strategy to avoid that the notation is influenced by the unavoidable limits of the available implementation platforms. This formal specification is currently demonstrated in an environment [28] particularly suitable to managing relational structures and their behavior thanks to the existence of software components devoted to the simulation and the check of properties of the obtained protocols. We are envisioning a new implementation that will exploit an agent based architecture and an Interoperability Language that expresses the various modes in which C-CMs and their components can be related [11] and a development environment particularly suitable to fully implement the interoperability requirements.

### 3. THE LAYERED STRUCTURE OF THE NOTATION

This section is devoted to the detailed presentation of the notation, starting from the  $\gamma$ -level.

#### 3.1 The $\gamma$ -level

The  $\gamma$ -level concerns the creation and modification of grammars for specifying the infinite variety of protocols necessary at the  $\beta$ -level.

The notation contains formalisms specific to the design of C-CMs:

- a) a set of **Basic Elements** which define a closed space of possibility from which each component of the grammar can be chosen.
- b) an **Interoperability Language**
- d) a **Malleability function** (Mall) and a **Linkability function** (Link) (see Section 3.1.3)

##### 3.1.1. The Interoperability Language

Since it will be used widely by the other components of the  $\gamma$ -level, we start with the presentation of the Interoperability Language.

The interoperability modes determine the basic primitives of the Interoperability Language that contains primitives taking the following general forms:

```
<Receiver> ! <message> <ack>
<Sender> ? <message> <ack>
```

with the following meaning, respectively: the element in whose specification the communication occurs sends to (!) / receives from (?) a <Receiver/ Sender> the content <message>. <ack> is a two-valued parameter: the default value 'must' requires an acknowledgment from the receiver, while the value 'may' does not.

Each <message> takes one of the following formats, where the operator <X>\* denotes finite sequence of items of sort X:

- tell* (<variable : value >\*) when the sender wants to communicate to the receiver the values of some variables;
- ask* (<variable >\*) when the sender wants to get information into some variables;
- perform* (<command>, <parameter>\*) when the sender wants the receiver performing the command with the given parameters; where <command> can be any of the following
  - update (<variable : value >\*) to assign values to variables;
  - read (<variable >\*) to read values into variables;
  - activate (<protocol>, <participant>\*) to start the execution of the <protocol> involving the mentioned <participant>\*. A <participant> is a Role, possibly augmented by its responsibility;
  - participate (<protocol>, <participant>\*) to involve the receiver in the execution of the <protocol>.

*synch* (<C-CM.Component.Action/Task>\*, <partial-order>) where each Action/Task belongs to a different C-CM. This message expresses that the Actions/Tasks have to be executed according to the specified partial order. Synch is characteristic to the control mode.

*over-write* (<items>\*, <new-items>\*) to enforce modifications on some items in the receiver. When the first group of parameters takes the value 'empty' the command corresponds to the creation of the second group of items. Over-write is characteristic to the over-rule mode.

Both in the case of a perform(read(...)) and an ask(...), the sender wants to get the value of the specified variables. The main difference is that in the first case the sender knows about the structure of the data owned by the receiver and therefore the command must terminate successfully; in the second case, the variables denote an information structure owned by the sender where the command deposits the asked values, if and when it terminates successfully. An analogous situation characterizes the differences between perform(update(...)) and tell(...).

The primitives of the Interoperability Language are used to describe the behavior of the various components of the notation at the  $\gamma$ -level. Each of them is represented in a frame-like notation whose attributes describe its characteristics and its interoperability capabilities.

### 3.1.2. Basic Elements

As described in section 2, there are two types of basic elements: the *objects* related to the dimensions of the articulation work (OAW); and the *formal structures* to express the relationships among the above objects. Every basic element has associated its (set of possible) *operational semantics*. Then, a grammar is created or modified by selecting its components from the set of basic elements together with their operational semantics. According to the definition of coordination mechanisms as an artifact together with the related procedures and conventions, the set of formal structures contains two type of items: the *relational structures* to describe the procedures and conventions and the *active artifacts* to represent the information contained in the artifact of the CM.

*Relational structures.* The field studies show that the possibility of representing relationships among entities is central to a notation for the design of CMs: e.g., causal relationships among tasks in workflows, part-of relations in classification schemes and so on. The literature provides several types of formalisms that serve exactly the purpose of expressing such relationships in an unambiguous way. In the current stage the proposal contains the following relational structures: Labelled-Graphs, as a very general purpose formalism to represent non-deterministic relationships; Labelled-AND/OR Graphs and different classes of Labelled-Petri-Nets, as means to represent causal relationships in presence of concurrency [2].

All of them are grounded on a sound mathematical theory providing algorithms for animation, simulation and analysis that can be exploited in the implementation of the primitives. All structures have associated labelling functions to express the interpretation of their constitutive elements, mainly in the set of Objects of Articulation Work (OAW).

To exemplify the notation, we show just a class of Labelled-Petri-Nets that emphasizes modularity, namely Superposed Automata nets [8], since it will be used in the working example:

**Labelled-SA-nets** ::= [name<sub>1</sub> : SM<sub>1</sub> || ... || name<sub>n</sub> : SM<sub>n</sub>]

where name<sub>i</sub> are names of objects that can be selected in the OAW (e.g., a Role); SM<sub>i</sub> are Labelled-State Machines where only transitions carry a label; and finally, || is the parallel composition based on the synchronization of sending/receiving messages.

We are not showing the SA-nets' semantics, rather we just emphasize that the semantics of any concurrent systems can be different in relation to the strategy adopted in executing the sets of concurrent actions: *full concurrency* semantics when all possible actions are executed in one shot; *step* semantics when any subset of possible actions is executed in one shot. This subset can be identified by means of arbitrary criteria (priority, common property of the labels, and so on); and finally *interleaving semantics*: when just one action at the time is selected and executed in a fully non-deterministic way. Details about this and formal definitions can be found in [29] in relation to Petri-net languages. These various semantics can be formulated equivalently in all formalisms representing concurrency.

*Active Artifacts.* The definition of CM assigns a relevant role to artifacts since they, due to their symbolic nature and standardized formats, contribute together with procedures and conventions to stipulate and mediate the articulation of distributed activities. Moreover, the definition of C-CM requires that such artifacts play

an *active role* in the articulation work since they must incorporate aspects of the protocol of a CM so that changes to the state of the mechanism induced by one role can be automatically conveyed by the artifact to other roles in an appropriate form as stipulated by the protocol.

In order to fulfil the above requirements, the notation should provide a structure able to represent structured information with the capability to communicate with its environment. This structure, called *active-artifact*, is defined as follows:

AA =

ATTRIBUTE NAME	ATTRIBUTE TYPE
name	identifier
access rights	<Role, type>*
content	data-frame
update/read requests	< X ? perform (update (info))>* < X ? perform (read (info))>*
accommodation	< condition -->    [X ! tell (info)]>*
awareness	< condition -->    [X ! tell (info), 'may']>* < condition & [X ? tell (info), 'may']--> function> *

where: type (update, read) and data-frame is a sequence of pairs <attribute-name : attribute-type> this latter belonging to the FoW. The conditions triggering *accommodation* and *awareness* communication can be based either on AA's internal states or on some event occurred in its behavior (e.g., an update by some component).

Accommodation is used to coordinate the AA with the other components of the C-CM it belongs to by means of messages where <ack> assumes value 'must'. Awareness describes the ability of the AA to be aware of its environment both in input and output. This means that the AA is able to send to its environment awareness messages and to react to awareness messages coming from outside possibly activating some internal function.

The behavior of an active-artifact can be described by two standard 'demons' that are in charge to manage the update/read requests and the accommodation/awareness communication, respectively, according to the Interoperability Language.

*Objects of Articulation Work.* The choice of the attributes of the OAW is based on the articulation model described in the introduction (Figure 1). Each attribute type carries the operational behavior associated with it. The operational semantics of the whole object is built up using these 'pieces of behavior'.

The attributes describing each object can be classified as follows:

- attributes identifying and describing the characteristics of the object: they vary from an object to another. They are collected in the single *description* attribute and not described here any further;
- attributes representing the relations with other objects;
- attributes making reference to policies. Their type is a pair <Role; policy> with the following meaning: The Role is responsible of the function evoked in the attribute name and accomplishes it by applying a specific policy. A policy is expressed as a set of rules or as a reference to another CM. For example, the assessment of the results of a Task has to be performed according to a predefined protocol; the declaration of a new organizational unit has to be authorized by a specific role [10].
- attributes representing the relationships with the Organizational Context where OAW are *managed, defined, assign-*

ned and adapted or the related *precepts* are defined. Precepts are a set of rules that determine the behavior of the objects (e.g. a role can be played only by actor with a certain skill and a certain age).

d) an attribute, called *awareness*, expressing the capability of each object to convey to its environment notification about changes of its internal states.

For example, the definition of the objects Role and Task is as follows.

**Role =**

ATTRIBUTE NAME	ATTRIBUTE TYPE
<b>description</b>	data-frame
<b>responsible of</b>	Resource*
<b>responsible of</b>	Task *
<b>precepts</b>	set of rules
<b>defined by</b>	<Role> ? over-write (empty, <new-Resource/ Task/ rule>*)
<b>adapted by</b>	<Role> ? over-write (<Resource/Task/rule>*, <new- Resource/Task/rule>*)
<b>awareness</b>	< condition -->    [X ! tell (info), 'may']>*< < condition & [X ? tell (info), 'may']--> function> *

A role is defined through a set of responsibilities of resources and tasks and a set of rules that are established in the Organizational Context by some enabled role. Then, the execution of the over-write commands defines a new set of responsibilities or new rules. Awareness can be described as for AA.

**Task =**

ATTRIBUTE-NAME	ATTRIBUTE-TYPE
<b>description</b>	data-frame
<b>in-triggers</b>	< X ? tell (info)> *
<b>precepts</b>	set of rules
<b>supervised by</b>	<Role; policy>
<b>criteria of accomplishment</b>	policy
<b>assessed by</b>	<Role ;policy>
<b>approved by</b>	<Role; policy>
<b>defined by</b>	<Role>? over-write (empty, new-attributes *)
<b>adapted by</b>	<Role> ? over-write (attribute*, new-attribute *)
<b>awareness</b>	< condition -->    [X ! tell (info), 'may']>*< < condition & [X ? tell (info), 'may']--> function> *

The last three attributes are described as in the previous case. In-triggers are notifications that 'must' be listen to. The remaining attributes specify how different roles have different duties that are performed according the associated policy: solving exceptions (supervise) or declaring that the task has been accomplished in a satisfactory way (assess and approve). The two latter attributes incorporate the idea of closed-loop that characterizes the proposal by Medina-Mora and associates [27] and at the same time they provide for a more flexible definition of the involved roles (e.g., the role who assesses in not necessarily the same as the role who ap-

proves). Recall that a policy can contain a reference to another C-CM, and then that these latter can be modified through the communication realizing over-rule.

The last OAW that will be used later on in the paper is Interaction. It contains four attributes: description, defined by, adapted by and awareness. They are described as in the previous objects. It is worth-mentioning that the 'description' contains as possible information the illocutionary point of the interaction. Illocutionary points take their values in Searle's taxonomy [34] and allow for the specification of Interactions with different associated pragmatics (offer, decline, etc. as in [10; 25; 39]).

We end this section by recalling that the current choice of Basic Elements is not definitive. In fact, the malleability and interoperability of C-CMs will possibly require additions to these basic elements if new coordination mechanisms cannot be represented by the notation, i.e. this notation should be extensible into unknown situations. The operation for doing this is not at the semantic level of articulation work, and is represented in our setting by the function ENRICH that makes the set of basic elements open to modifications: ENRICH accesses the programming environment where the new basic elements can be defined and imported in the notation.

### 3.1.3. Malleability and Linkability

Besides the basic elements illustrated above, at the  $\gamma$ -level the notation contains a malleability function (Mall) that is used when a new grammar has to be defined or an existing one modified.

Mall takes a grammar as an argument: Mall (grammar) = new-grammar. If the argument is the empty grammar, then Mall denotes a creation; otherwise, it denotes a modification. The function Mall is defined along all components of the grammar. The creation of a new grammar starts by the definition of the production having as left-hand-side the initial symbol START; the modification of a grammar operates on the existing productions. All the grammars already defined are collected into a set called: Grammar-set.

As an example, let us consider the definition of a grammar, called CONV\_GR, for the construction of different types of conversation models [39] (the protocols) that are traditionally described by means of a Labelled Graph. Then the first step of the definition of CONV\_GR assigns to the symbol START an L-Graph whose arcs are labelled in the set of the interactions. This is realized by making available a framework where L-Graphs can be edited and the related arcs labelled in the set of the Interactions. The arc labelling function is not arbitrary. Indeed, it has to follow some semantic constraints: for example, the fact that an *accept, counteroffer...* must follow a *request/offer*. This property can be represented in the notation by suitable predicates that are defined together with the grammar.

As a second example, let us consider the definition of a grammar called WF\_GR for the construction of workflows that the designer wants to represent by a formalism describing distributed states and actions. Then, the designer chooses to base the description on SA-nets.

WF\_GR accesses a framework where Labelled SA-nets can be edited and their transitions are labelled either as a task or an interaction. The names of the constituting State machines are defined as the name of a Role extended by the acronym of the current CM.

Once a grammar has been defined, then it can be modified through the Mall function, on some of its components, by using alternative basic elements and/or alternative semantics. For example, Mall(Relational-Structures) changes the set of structures used in the grammar. E.g., in the CONV\_GR one can define the START symbol as:

START ::= CONV\_L-Graph / CONV\_L-Petri-net

where CONV\_L-Petri-net is a Petri-net whose transitions are labelled in the Interactions, in order to allow for the definition of multi-agent distributed interactions in addition to the well-known conversations.

The expressive power can be changed acting also on the labelling functions by modifying the type of the objects constituting the labelling sets. For example, in the WF\_GR one can allow only tasks as labels of the transitions and change the labelling of names (Mall(L)) from Roles to generic labels. Or one can construct another grammar called WF/2\_GR by associating to the SA-net transitions labels of type Actions (both performative actions and interactions).

It is possible to modify also the semantics of a component of the grammar and not the component itself. For example, in the case of relational structures modelling concurrency one could select one of the alternative associated semantics.

The Linkability function allows one to build a composed CM from two or more existing ones. This composition is defined by specifying the interface each C-CM presents to the linked C-CMs. The interface establishes which information can be mutually accessed and communicated by the involved mechanisms. The constituents of the interface are:

1. the access rights of other C-CMs on the Active Artifact, namely update and read capabilities, possibly restricted to some specific slots of the data-frame of the AAs.
2. the primitives to express accommodation, awareness (as in the case of AA) and over-rule (as in the case of OAWs) among the C-CMs.
3. the primitives expressing references among the C-CMs.
4. the primitives containing the control message: `synch(<C-CM.Component.Action/Task>*, <partial-order>)`.

The presently considered causal relationships are: sequence and concurrency.

Then the interface is a sort of wrapper [16] that monitors the behavior of the C-CM to handle the additional accommodation and awareness messages and collects and manages the additional communication needs derived from the linking of different C-CMs. The interface is the place where CMs can be made tolerant to the modifications of the CMs linked to them. In fact, the interfaces can be conceived of as agents specialized not only in the management of the communication but also in the 'translation' of the incoming/outcoming information in a format that can be interpreted by the CM holding the interface. This aspect will be a central issue of the next agent-based implementation. If the translation is not possible, then the intrerfaces will notify it and support the activation of the suitable negotiation/recovery protocols.

Then the linking function takes the following form:

Link( $p_1$ ,  $p_2$ , Interface<sub>1</sub>, Interface<sub>2</sub>)

where  $p_1$ ,  $p_2$  are the protocols modeling the C-CMs to be linked and Interface <sub>$i$</sub>  ( $i = 1, 2$ ) are arguments formalizing the interface described above. An example of definition of interface, and consequently of the linking function, is given in the following section where the definition of protocols is described.

### 3.2 The $\beta$ -level

At this level, the grammars contained in the Grammar-set are 'visible' to the users so that they can define the protocols they need. In order to facilitate this the notation at the  $\beta$ -level is mainly

constituted by a *set of primitives* supporting the manipulation of protocols. At the  $\beta$ -level the notation contains:

1) a Grammar-set, containing the grammars constructed at the  $\gamma$ -level and the related primitive **access**(Grammar-set). If we consider the grammars defined in the previous section, then Grammar-set = {WF\_GR, CONV\_GR, WF/2\_GR}.

2) a primitive to define the protocols using the grammars of the previous point: namely, **define**-protocol(X) where X is a protocol name.

3) a primitive to modify protocols: namely, **modify**-protocol(X, modification-type) where X is a protocol name.

The define and modify primitives provide some support for checking the correctness of the new protocol against some predefined property by exploiting checking techniques provided by the formal structures made available by the notation. This idea is still present for example in DOMINO [24] where Petri-nets algorithms are exploited [5] and in the proposal contained in [13] where a proof of correctness of a modification of ICN [14] exploits the theory of graph grammars [12] as ICN are based on AND/OR-Graphs.

Define and modify can be 'implemented' as a communication sent from the user interface to the current protocol, namely Curr Prot. ! perform(over-write (suitable-parameter\*))

4) **animate**(X) and **simulate**(X) provide the user with the possibility of 'playing' with the protocol to perform a sort of test by exploiting the operational semantics associated to the formalism underlying the protocols. Simulation is an animation with the additional computation of predefined parameters expressing good behaviors.

Animate and simulate can be 'implemented' as a communication sent from the user interface to the current protocol, namely Curr Prot. ! perform(activate (suitable-parameter\*))

5) **build**(history(X)) and **access**(history(X)) are primitives providing the classical support to the management of the various versions of a protocol.

The best way of illustrating the  $\beta$ -level is by means of an example of protocol definition taken from a field study reported in [7]: the Bug Handling protocol within the software development process

The roles that are relevant for registering, diagnosing, and correcting software bugs are: the tester, involved in the actual testing of the software in the S4000 instrument; the spec-team, a group of three software designers being responsible for diagnosing the bugs and deciding how to handle the correction of the bugs; the software designers, being in charge of the correction. All software designers are responsible for one or more software module. Since a bug is always related to a specific module, one of the designers will always be responsible for correcting a specific bug. The platform master, one of the designers in the project, responsible for verifying the corrections made by the designers.

Initials: Instrument:	Report no:	tester
Date:		spec-team
Description:		tester
Classification: 1) Catastrophic 2) Essential 3) Cosmetic		spec-team
Involved modules: Responsible designer:		spec-team.
Estimated time:		
Date of change: Time spend:		
Tested date:		responsible designer
<input type="checkbox"/> Periodic error - presumed corrected		
Accepted by: Date:		
To be: 1) Rejected 2) Postponed 3) Accepted		spec-team
Software classification (1-5): ____		
Platform:		
Description of corrections:		
Modified applications:		responsible designer
Modified files:		

Figure 3. The bug report form.

The artifact is the bug form report (described in Figure 3 together with additional information about which role is supposed to fill in which field following the conventions stipulated in the group).

The data frame of the corresponding active artifact models the information conveyed by the bug report form (hereafter BR): its formalization is omitted.

Let us suppose that the designer decides to exploit the expressive power of the WF\_GR grammar described above. Then the Bug Handling protocol (hereafter BH) can be defined componentwise leading to the description of Figure 4 which contains the BR as active artifact and a set of elemental protocols describing the behavior of the various roles within the BH.

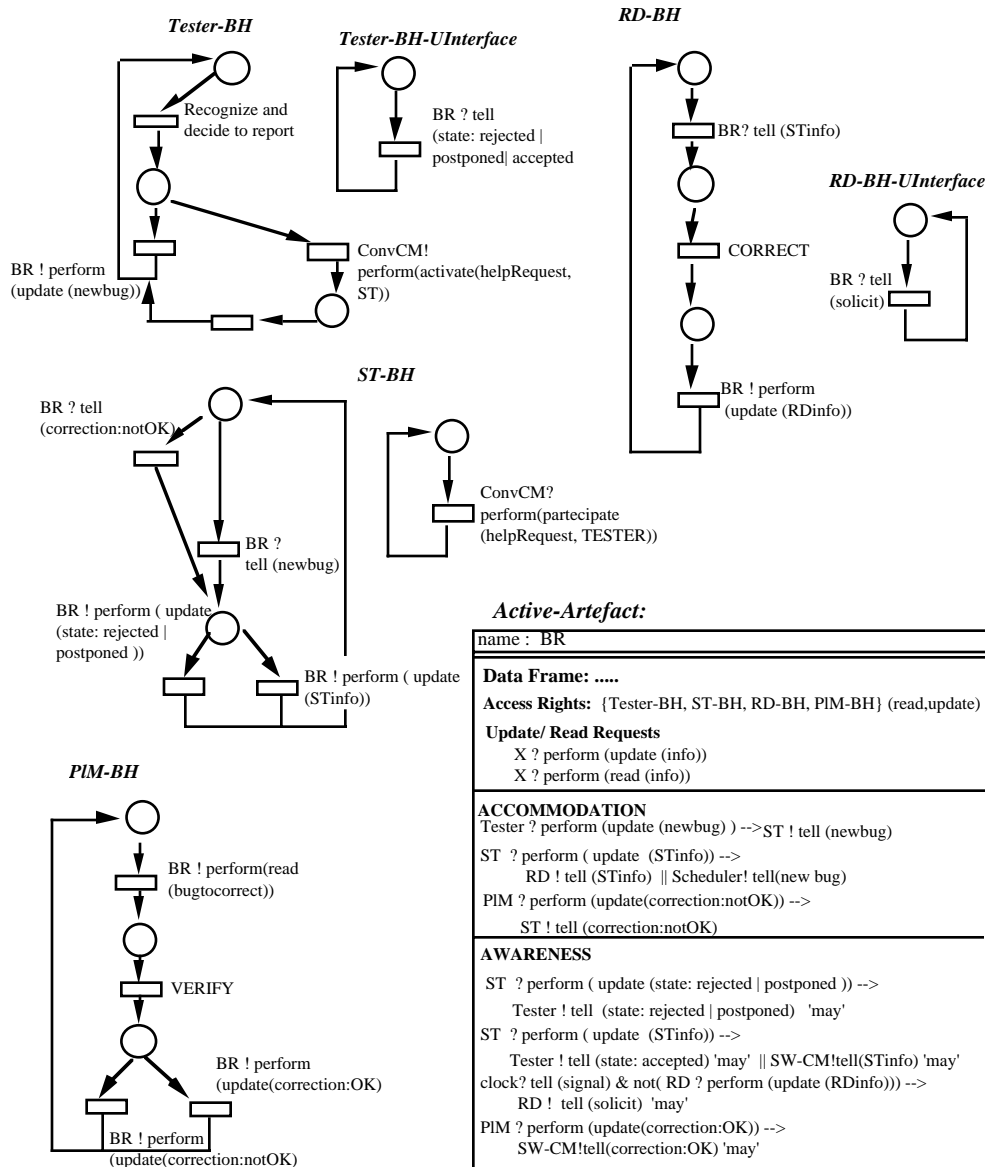


Figure 4. The BH Protocol.

In the first part of the BR, the access rights are specified. This means that all the elemental protocols of the BH can send to BR a request to perform both a read and an update, with the specified form (X stands for one of the roles indicated in the access rights).

In the second part, the accommodation of the BR and the elemental protocols is described. For example, let us consider when the spec-team requires the BR to update its values (BR! perform(update (STInfo)) where 'STInfo' stands for a list of the type <attribute name: attribute value>\*, indicating which characteristics of the BR the spec-team is conveying). On this event the BR reacts with a message to the responsible designer (RD! tell (STInfo)), communicating that a new bug has to be processed and conveying all the information provided by the spec-team.

The considered triggering event appears in the section describing awareness too. In fact, when the spec-team accepts a bug, the BR informs the tester on the evolution of the bugs s/he reported. This is only a sort of notification that does not impact on the behavior of the tester and it is processed by her/his UserInterface.

The transition labelled *ConvCM! perform (activate(helpRequest, ST))* in the specification of the tester behavior is an example of reference to another CM. With this the tester requires the Conversation CM to activate the protocol 'help request' (a predefined conversation protocol) involving the spec-team as partner. In the specification of the spec-team is indicated that s/he can, at every moment, be involved in a conversation with a tester asking for help, i.e. receiving from the Conversation CM a message whose content is *perform (participate (helpRequest, tester))*.

Let us now consider the relations of the BH with its external world represented by the C-CMs interoperating with BH, namely a Scheduler (of the software activities), a SW-CM (managing the software decomposition into modules), and finally the SW Process Manager (SW-P.M), that is, the component of the C-CM SW Process describing the behavior of the Manager responsible for the task devoted to changing the bug report process. Then

### BH-Interface=

<b>name</b>	BH
<b>accommodation</b>	BR:ST ? perform (update (STinfo)) -> Scheduler! tell(new-bug)
<b>awareness</b>	BR:ST ? perform (update (STinfo))->SW-CM!tell (STinfo) 'may' BR:PIM?perform(update(correction: OK))->SW-CM!tell ((correction: OK)) 'may'
<b>reference to</b>	tester-BH:ConvCM! perform(activate(helpRequest, ST-BH)) ST-BH:ConvCM?perform (participate (helpRequest, tester-BH) *)
<b>referenced by</b>	none
<b>access rights</b>	none
<b>control</b>	Scheduler ? synch((Scheduler...AssignPIM, BH.PIM-BH. (BR ! read(bug-to-correct))), sequence) Scheduler ! synch ((Scheduler..AllocateResources, BH.RD-BH. (correct)),sequence)
<b>defined by</b>	SW-P.M?over-write (empty, "see figure 4")
<b>adapted by</b>	SW-P.M ? over-write(BR, tester, PIM, new-BR, new- tester, new-PIM)

The meaning of the various slots are as follows:

*accommodation*: When the Spec-team accepts a bug, the BH must tell the Scheduler that a new correction must be considered during the planning.

*awareness*: Moreover when the new bug is accepted, the BH notifies the SW-CM, so that the latter is aware that a bug has been detected within a specific module. The same happens when the bug is corrected

*reference*: to express that BH makes use of the Conv-CM.

The above slots are automatically derived from the definition of the protocol given by the designer. The following ones have to be explicitly given by the designer at the invocation of the Linking function:

*access rights*: No CM has access rights on the AA of the BH.

*control*: In the first communication the BH is passive, in the sense that the PIM is suspended until the Scheduler performs 'AssignPIM' and informs the BH. In the second one the BH must require the activation of 'Allocate Resources' before 'correct' can be performed by the RD.

*defined/adapted by*: the BH is prepared to receive from the SW-P.M two types of messages: one for its definition and one for its adaptation (In the interface we give only an example of possible over-ruling).

### 3.3 $\alpha$ - level

At this level, the protocols contained in the Protocol-set are 'visible' to the users so that they can define the instances they need. This level is quite standard in many applications: what characterizes the proposed notation is the set of primitives that are available to manage the instances of the protocols (local control). Accordingly, the notation at the  $\alpha$ -level is constituted by (let Y be a protocol name and X be a Y-Instances name):

1) a Protocol-set, containing the protocols constructed at the  $\beta$ -level and the related primitive **access**(Protocol-set).

2) a primitive to define instances using the protocols of the previous point: namely, **define**-instance(X, Y). At the  $\alpha$ -level the relationships with the FoW concern the attributes values.

3) a primitive to activate the instance, that is, to put the current C-CM at work in a specific circumstance: namely, **activate**(X). This primitive is based on the operational semantics of the protocol X is an instance of.

Besides these two primitives, which are standard in many applications, there are two primitives allowing one to modify the behavior of an instance in two ways:

4) **modify**-instance(X) allows for structural modifications of the instance: for example, if its source protocol exploits graphs as formal structure, **modify**(X) allows the insertion/deletion of arcs and nodes and/or the change of their labelling functions. These modifications are possible also when the instance has been activated and affect just the current instance while the source protocol remains unchanged. A similar functionality is provided for example in EGRET [20].

5) **enforce**(X, new-configuration) can be used when the instance has been activated. Its invocation allows for an instance behavior to proceed from a new configuration with respect to the current one. For example, if we consider a graph-based formalism, the configuration is made of the current node; in the case of Petri-nets based formalisms, the configurations are the markings; in the case of an active-artifact the configuration is the current set of values. Then, the primitive allows one to change node, marking and values, respectively, in a way that is independent of the previous configurations.

The activate, modify and enforce primitives can be 'implemented' as a communication sent from the user interface to the current instance.

6) **build**(history(X)) and **access**(history(X)) are primitives that record and make available the various steps the behavior of the instance went through after its activation. Basically, they handle the history of the execution of X.

7) the last primitive **makePermanent**(X, new-name) allows one to transform an instance into a permanent protocol that will be referenced by the new-name. This operation makes sense after the activation of the primitive allowing for structural modifications that are becoming recurrent so that they can be made permanently available for future uses. A similar functionality is provided in, e.g., EGRET [20].

## CONCLUSIONS

At the present stage of our research of coordination mechanisms, a notation for constructing computational coordination mechanisms at the semantic level of articulation work has been developed and specified formally, and it has been demonstrated that this notation in principle makes it possible to construct coordination mechanisms that are malleable to any degree deemed appropriate for any particular setting. The notation is currently being implemented concurrently with the design of a number of experimental C-CMs for CSCW applications for software engineering.

## ACKNOWLEDGMENTS

The research reported in this paper is supported by the European Union's ESPRIT Basic Research project COMIC (Action 6225) and by the Danish Research Council for the Natural Sciences. Our thanks are due to our partners in the project, in particular to Peter

Carstensen and Carsten Sørensen, and to the anonymous reviewers of the initial submission, for their suggestions and comments.

## REFERENCES

1. Andersen, Hans H. K.: "Classification schemes: Supporting articulation work in technical documentation," in H. Albrechtsen (ed.) *ISKO '94. Knowledge Organisation and Quality Management, Copenhagen, Denmark, June 21-24, 1994*, 1994.
2. Bernardinello, Luca, and Fiorella De Cindio: "A Survey of Basic Net Models and Modular Net Classes," in G. Rozenberg (ed.) *Advances in Petri Nets 92, LNCS 609*, Springer-Verlag, Berlin, Germany, 1992, pp. 304-351.
3. Bogia, Douglas P., William J. Tolone, Celsina Bignoli, and Simon M. Kaplan: "Issues in the Design of Collaborative Systems: Lessons from ConversationBuilder," in D. Shapiro, M. Tauber and R. Traünmüller (eds.): *The Design of Computer-Supported Cooperative Work and Groupware Systems*, Elsevier Science, Amsterdam, 1994. - Forthcoming.
4. Bowker, Geoffrey, and Susan Leigh Star: "Situations vs. Standards in Long-Term, Wide-Scale Decision-Making: The Case of the International Classification of Diseases," in J. F. Nunamaker, Jr. and R. H. Sprague, Jr. (eds.): *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences, Kauai, Hawaii, January 7-11, 1991*, IEEE Computer Society Press, 1991, vol. IV.
5. Brauer, W., W. Reisig, and G. Rozenberg (eds.): *Petri Nets: Central Models and Their Properties*, vol. 254, Springer-Verlag, Berlin, Germany, 1987.
6. Carstensen, Peter, Tuomo Tuikka, and Carsten Sørensen: "Are We Done Now? Towards Requirements for Computer Supported Cooperative Software Testing," in P. Kerola, A. Juustila and J. Järvinen (eds.): *17th IRIS Seminar, Syöte, Finland, 6-9 August, 1994*, Oulu University, 1994, vol. 1, pp. 424-451.
7. Carstensen, Peter H., Carsten Sørensen, and Henrik Borstrøm: "Two is Fine, Four is a Mess: Reducing Complexity of Articulation Work in Manufacturing," in *COOP '95. International Workshop on the Design of Cooperative Systems, Antibes-Juan-les-Pins, France, 25-27 January 1995*, INRIA Sophia Antipolis, France, 1995, pp. 314-333.
8. De Cindio, Fiorella, Giorgio De Michelis, Lucia Pomello, and Carla Simone: "Superposed Automata Nets," in G. Girault and W. Reisig (eds.): *Application and Theory of Petri Nets, IFB 52*, Springer-Verlag, Berlin, Germany, 1982.
9. Degani, Asaf, and Earl L. Wiener: *Human Factors of Flight-Deck Checklists: The Normal Checklist*, National Aeronautics and Space Administration, Ames Research Center, Moffett Field, California, May, 1990. [NASA Contractor Report 177549; Contract NCC2-377].
10. Divitini, Monica, and Carla Simone: "A Prototype for Providing Users with the Contexts of Cooperation," in R. Oppermann, S. Bagnara and D. Benyon (eds.): *ECCE7, Bonn, Germany, September 5-8, 1994*, GMD, 1994, pp. 253-270.
11. Divitini, Monica, Carla Simone, Kjeld Schmidt, and Peter Carstensen: *A multi-agent approach to the design of coordination mechanisms*, Working Papers in Cognitive Science and HCI, Roskilde University, DK-4000 Roskilde, Denmark, 1995. [WPCS-95-5].
12. Ehrig, H., M. Nagl, and G. Rozenberg (eds.): *Graph Grammars and Their Application to Computer Science (2nd International Workshop)*, Springer-Verlag, Berlin, 1983.
13. Ellis, Clarence A., Karim Keddara, and Grzegorz Rozenberg: "Dynamic Change Within Workflow Systems," in *COOCS '95. Conference on Organizational Computing Systems, Milpitas, California, August 13-16, 1995*, 1995. - Forthcoming.
14. Ellis, Clarence A., and Gary J. Nutt: "Office Information Systems and Computer Science," *Computing Surveys*, vol. 12, no. 1, March 1980, pp. 27-60.
15. Flores, Fernando, Michael Graves, Brad Hartfield, and Terry Winograd: "Computer Systems and the Design of Organizational Interaction," *ACM Transactions on Office Information Systems*, vol. 6, no. 2, April 1988, pp. 153-172.
16. Genesereth, Michael R., and Steven P. Ketchpel: "Software Agents," *Communications of the ACM*, vol. 37, no. 7, July 1994, pp. 48-53, 147.
17. Gerson, Elihu M., and Susan Leigh Star: "Analyzing Due Process in the Workplace," *ACM Transactions on Office Information Systems*, vol. 4, no. 3, July 1986, pp. 257-270.
18. Harper, Richard R., John A. Hughes, and Dan Z. Shapiro: *The Functionality of Flight Strips in ATC Work. The report for the Civil Aviation Authority*, Lancaster Sociotechnics Group, Department of Sociology, Lancaster University, January, 1989.
19. Heath, Christian, and Paul Luff: "Collaboration and Control. Crisis Management and Multimedia Technology in London Underground Control Rooms," *Computer Supported Cooperative Work (CSCW). An International Journal*, vol. 1, no. 1-2, 1992, pp. 69-94.
20. Johnson, Philip: "Supporting Exploratory CSCW with the EGRET Framework," in J. Turner and R. Kraut (eds.): *CSCW '92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992*, ACM Press, New York, 1992, pp. 298-305.
21. Kaplan, Simon M., William J. Tolone, Douglas P. Bogia, and Celsina Bignoli: "Flexible, Active Support for Collaborative Work with Conversation Builder," in J. Turner and R. Kraut (eds.): *CSCW '92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992*, ACM Press, New York, 1992, pp. 378-385.
22. Kreifelts, Thomas, Elke Hinrichs, Karl-Heinz Klein, Peter Seuffert, and Gerd Woetzel: "Experiences with the DOMINO Office Procedure System," in L. Bannon, M. Robinson and K. Schmidt (eds.): *ECSCW '91. Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, Kluwer Academic Publishers, Amsterdam, 1991, pp. 117-130.
23. Kreifelts, Thomas, Frank Victor, Gerd Woetzel, and Michael Weitass: "A Design Tools for Autonomous Agents," in J. M. Bowers and S. D. Benford (eds.): *Studies in Computer Supported Cooperative Work. Theory, Practice and Design*, North-Holland, Amsterdam etc., 1991, pp. 131-144.
24. Kreifelts, T., and G. Woetzel: "Distribution and Error Handling in an Office Procedure System," in G. Bracchi and D. Tsichritzis (eds.): *Office Systems: Methods and Tools*, Elsevier Science Publishers B.V., North-Holland, 1987, pp. 197-208.
25. Labrou, Yannis, and Tim Finin: "A semantics approach for KQML - a general purpose communication language for software agents," in *3rd International Conference on*

- Information and Knowledge Management, CIKM'94, November, 1994, 1994. - Forthcoming.*
26. Malone, Thomas W., Kum-Yew Lai, and Christopher Fry: "Experiments with Oval: A Radically Tailorable Tool for Cooperative Work," in J. Turner and R. Kraut (eds.): *CSCW '92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992*, ACM Press, New York, 1992, pp. 289-297.
  27. Medina-Mora, Raul, Terry Winograd, Rodrigo Flores, and Fernando Flores: "The Action Workflow Approach to Workflow Management Technology," in J. Turner and R. Kraut (eds.): *CSCW '92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992*, ACM Press, New York, 1992, pp. 281-288.
  28. Meta Software: *DesignML Reference Manual*, v. 1.0, Meta Software Corporation, Cambridge, MA, 1990.
  29. Pomello, Lucia, Grzegorz Rozenberg, and Carla Simone: "A Survey of Equivalence Notions for Petri Net based Systems," in G. Rozenberg (ed.) *Advances in Petri Nets 92, LNCS 609*, Springer-Verlag, Berlin, 1992, pp. 410-472.
  30. Schmidt, Kjeld: "Riding a Tiger, or Computer Supported Cooperative Work," in L. Bannon, M. Robinson and K. Schmidt (eds.): *ECSCW '91. Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, Kluwer Academic Publishers, Amsterdam, 1991, pp. 1-16.
  31. Schmidt, Kjeld: *Modes and Mechanisms of Interaction in Cooperative Work*, Risø National Laboratory, P.O. Box 49, DK-4000 Roskilde, Denmark, 1994. [Risø-R-666(EN)].
  32. Schmidt, Kjeld, and Carla Simone: "Mechanisms of Interaction: An Approach to CSCW Systems Design," in *COOP '95. International Workshop on the Design of Cooperative Systems, Antibes-Juan-les-Pins, France, 25-27 January 1995*, INRIA Sophia Antipolis, France, 1995, pp. 56-75.
  33. Schmidt, Kjeld, Carla Simone, Monica Divitini, Peter Carstensen, and Carsten Sørensen: *A 'contrat sociale' for CSCW systems: Supporting interoperability of computational coordination mechanisms*, Working Papers in Cognitive Science and HCI, Roskilde University, DK-4000 Roskilde, Denmark, 1995. [WPCS-95-7].
  34. Searle, John R.: "A Taxonomy of Illocutionary Acts," in K. Gunderson (ed.) *Language, Mind, and Knowledge*, vol. VII, University of Minnesota Press, Minneapolis, 1975, pp. 344-369.
  35. Strauss, Anselm: "Work and the Division of Labor," *The Sociological Quarterly*, vol. 26, no. 1, 1985, pp. 1-19.
  36. Suchman, Lucy A.: "Office Procedures as Practical Action: Models of Work and System Design," *ACM Transactions on Office Information Systems*, vol. 1, no. 4, October 1983, pp. 320-328.
  37. Suchman, Lucy A.: *Plans and situated actions. The problem of human-machine communication*, Cambridge University Press, Cambridge, 1987.
  38. Suchman, Lucy A., and Eleanor Wynn: "Procedures and Problems in the Office," *Office: Technology and People*, vol. 2, 1984, pp. 133-154.
  39. Winograd, Terry, and Fernando Flores: *Understanding Computers and Cognition: A New Foundation for Design*, Ablex Publishing Corp., Norwood, New Jersey, 1986.