

Taking the distributed nature of cooperative work seriously

Carla Simone¹, Kjeld Schmidt²

¹Dept. of Computer Sciences, University of Torino, Corso Svizzera 185,
I-10148 Torino, Italy. Email: *simone@di.unito.it*

²Center for Tele-Information, Technical University of Denmark,
DK-2800 Lyngby, Denmark. Email: *schmidt@cti.dtu.dk*

Abstract

For CSCW facilities to be effective and viable, the inherently distributed nature of cooperative work must match by a radically distributed environment. On the basis of a scenario derived from field studies, the paper describes a CSCW environment which supports the distributed construction and use of malleable and linkable coordination mechanisms.

Cooperative work is constituted by mutual interdependence of activities. However, by involving multiple actors cooperative work is inherently and inexorably distributed. No agent is all-knowing and all-powerful. Actors must act and interact on the basis of partial knowledge and are, accordingly, partially autonomous in their work.

The myriad interdependent and yet distributed activities must be articulated to prevent the cooperative effort from 'trashing around chaotically', to use Waldrop's apt expression [16, p. 109]. They must be coordinated, aligned, meshed, etc. The obvious and fundamental way to do that is to facilitate mutual awareness among actors, for instance, by having actors working in the same room or by providing some multi-media emulation of a 'shared space'.

However, task interdependencies are often of an order of complexity where the provision of facilities for mutual awareness and ad hoc interactions is insufficient. Other means are required which make task interdependencies tractable. We call such means coordination mechanisms [12]. A coordination mechanism is, simply put, a coordinative protocol with an accompanying artifact, such as, for instance, a standard operating procedure supported by a certain form.

A coordination mechanism offers a 'precomputation' [11] of task interdependencies and is thereby instrumental in reducing the space of possibilities facing a competent actor in a given situation. However, the distributed nature of cooperative work is not eradicated by coordination mechanisms. They are merely local and temporary closures which assist actors in managing an otherwise overwhelming complexity. They are used in a distributed way

and evolve through a process of local adaptations. The challenge is thus to provide a CSCW environment which supports the construction and use of computational coordination mechanisms which are robust in view of the distributed nature of cooperative work.

1. A real-world scenario

Consider a team of software designers who are engaged in developing a large software system.

The design task poses a major challenge to them as they have no established routines and conventions for handling a cooperative design effort of this size and for ensuring that the individual designers' contributions can be integrated.

In order to manage the complexity of such a project, the software designers have devised and introduced a number of procedures, conventions and related artifacts to help them coordinate their cooperative effort. For the sake of simplicity, we will limit the scenario to four artifacts and associated procedures and conventions used in the context of software testing.¹

The Project Schedule. A Project Schedule was introduced, as a paper based artifact, to capture and display the relationships between actors, responsibilities, tasks, and deadlines. The latter are mainly related to the particular notion of the 'platform period' within which all software designers are coding and integrating their bits and pieces. For each such platform period, one of the designers is appointed Platform Master which implies that he or she will be responsible for updating information on changes made to the software and for ensuring that the software is tested and corrected before it is released. Moreover, before the software is released as a 'platform' for further development, the Platform Master updates the project schedule with revised plans and tasks for the next period.

The Directory. The directory structure of the software module repository provides a classification scheme for enabling distributed storage and retrieval of tested software

¹ Our scenario is based upon field studies at Foss Electric in Denmark. In these field studies different aspects of a major design effort, the 'S4000 project', were investigated. The field study findings have been reported elsewhere [e.g., 2].

modules. In addition, it provides a common standardized conceptualization of essential aspects of the field of work, namely the software architecture. All designers can relate to this when communicating and coordinating their activities.

The Bug Report Form. The Bug Report Form is developed to handle the process of software testing and as an artifact it is a simple paper form. When a new bug is detected by anyone involved in testing the software, a new bug report form is initiated and filled-in with a preliminary description and diagnosis of the problem. The designer responsible for diagnosing and classifying all reported bugs then determines the presumably culpable module and, by implication, the designer responsible for that module and specifies the platform integration period by which the bug should be corrected, and classifies the bug according to its perceived severity (as seen from a software reliability perspective). Finally, each designer is responsible for fixing the problems and reporting back to the Platform Master, i.e., the designer responsible for the next software module integration and verification period.

The Binder. All bug forms are filed in a public repository ('the binder') which is organized according to the status of the bugs. The forms collected in the binder are successively re-classified and re-filed according to decisions made by the designers, messages concerning specific bugs, results from the verification of the reported corrections from the platform integration period, etc.

From time to time, the software designers will experience situations where they need to modify a protocol they have devised and adopted as it does provide adequate coordination. These modifications may be merely local and temporary. For example, a tester will occasionally inform a software designer directly of a detected bug, without filling-in a bug report form and initiating a new instance of the protocol. On the other hand, the actors may want to change the protocol for good. The software designers may, for example, introduce the role of a project manager in the protocol.

It is obvious from our description that the different protocols interact in the sense that information or event notifications generated locally propagate across protocols, in order to meet the integration objectives and to solve critical situations.

2. Analysis

The assemblage of protocols described in this scenario could, of course, be modelled as an integrated 'workflow'. This would, however, be utterly misleading in view of the distributed nature of cooperative work.

First of all, each of the protocols is devised to serve specific purposes in the setting. They address a very narrow set of coordinative activities. They are local and temporary closures. Of course, the protocols intersect at various points and therefore need to be aligned by the actors in the course of their work. Anyway, while the protocols are interlaced, they are only loosely coupled. Because of that,

each protocol can be constructed relatively independently of the others.

Thus, in designing a protocol for a specific purpose, actors do not need to have an overview of the totality of work processes in the setting at large. In fact, in settings such as our scenario it is unlikely that anybody should have such an overview. Hence, because such protocols can be designed and introduced to serve limited purposes, the array of interlaced protocols of the work arrangement at large can be designed and maintained in a distributed manner. It evolves through a process of local actions and interactions.

Furthermore, there is a specific and crucial relationship between protocol and artifact which is absent from the concept of workflows.

(1) The protocol is objectified by the artifact which thus gives a certain permanence to the protocol. That is, it conveys the stipulations of the protocol in a situation-independent manner.

(2) The artifact serves as an *intermediary* between actors in that it mediates information about state changes to the protocol as it is being executed. In the case of the bug report, for example, the state of the artifact changes according to the changing state of the protocol. As in the case of any workflow, the form is transferred from one actor to another and this change of location transfers to the particular actor the specific responsibility of taking a specific action on this particular bug. However, at each step in the execution of the protocol, the form is annotated and the thereby updated form retains and conveys this change to the state of the protocol to the other actors. That is, a change to the state of the protocol induced by one actor (a tester reporting a bug, for example) is conveyed to other actors by means of a visible and durable change to the artifact. In so far, the artifact can be said to provide a 'shared space', to use a popular CSCW term, albeit with important restrictions: it is a highly restricted space with a structure that reflects salient features of the protocol, and it is only 'shared' locally, within the context of the particular instantiation of the protocol.

(3) It is this close relationship between protocol and artifact which makes local control of the execution and evolution of the protocol feasible. By objectifying the protocol, the artifact also delimits the scope of the protocol in a tangible way. The very materiality of the artifact thus facilitates a distributed, incremental evolution of the entire population of protocols. Moreover, also due to its materiality the artifact facilitates a (restricted as well as focused) mutual awareness among actors which, *inter alia*, enables them to decide whether or not to deviate from the protocol in the face of contingencies.

In short, the artifact plays a crucial role for the protocol by objectifying it, by representing salient features, by mediating the state of the protocol, etc. The artifact and the protocol work 'hand in glove'.

3. Computational coordination mechanisms

We have elsewhere [12] defined a computational coordination mechanism as a software device in which the artifact as well as aspects of the protocol of a coordination mechanism are incorporated in a computational form, so that changes to the state of the protocol induced by one actor are conveyed by the computational artifact to other actors in accordance with to the protocol.

A computational coordination mechanism should meet the following requirements:

Firstly and indispensably, coordination mechanisms must be malleable. Actors should be able to define the protocol of a new computational coordination mechanism, as they indeed did in our scenario. It should also be possible for actors to redefine it by making lasting modification to it, so as to be able to meet changing organizational requirements, for instance in order to introduce a new role in the bug report protocol. Furthermore, actors must be able to control the execution of the protocol and make local and temporary modifications to its behavior to cope with unforeseen contingencies. For example, in the scenario we observed that a tester will occasionally inform a software designer directly of a detected bug, without filling-in a bug report form and initiating a new instance of the protocol. It should be feasible to circumvent the protocol and yet ensure the reporting of bugs and corrections. Accordingly, in order for actors to be able to define, specify, and control the execution of the mechanism, the protocol must be visible to actors at the semantic level of articulation work, i.e., it must be accessible as well as expressed in terms that are meaningful to competent members of the cooperative work arrangement. Finally, to allow for incomplete initial specification of the protocol, it should be possible to specify the behavior of the computational coordination mechanism incrementally, while it is being executed.

Furthermore, the evolution of coordination mechanisms through continual adaptation is itself a distributed effort. That is, a CSCW environment must support local modifications to the protocol and, accordingly, local control of the propagation of changes.

Last, but not least, a computational coordination mechanism should be constructed in such a way that it can be linked to other coordination mechanisms in the wider setting.

In short, to be able to construct computational coordination mechanisms that meet these requirements, we need a highly versatile environment:

- Versatility in terms of control strategies, e.g., procedures vs. classification schemes, explicit prescriptions vs. implicit constraints.
- Versatility in terms of protocol modelling, e.g., document flow, control flow, communication flow: the artifacts are of different nature and pose quite different modeling problems, as do the various protocols.

- Versatility in terms of composition: we must be able to construct different ways of interaction between artifact and protocol.
- Versatility in terms of interaction between mechanisms, i.e., support of different kinds of linking.
- Support of distributed control of executions and evolution of mechanisms, including local control of the propagation of changes.

The Ariadne environment is designed to meet these requirements.

4. The Ariadne environment

Ariadne is an environment specifically dedicated to the construction, maintenance, execution, and mutual linking of computational coordination mechanisms governing cooperative activities with respect to the target applications.

Ariadne contains a notation, i.e., a system of symbols and rules for their application, as well as the primitives necessary for the use of the notation.

4.1. The notation

As for the modeling of coordination mechanisms and the protocols they are based on, the required versatility is achieved through the identification of categories that are flexibly composed via a set of predefined relational structures. Hence the crucial point is to determine a repertory of categories, at the semantic levels of articulation work, by means of which coordinative protocols can be expressed. The proposed set of categories was derived from the studies of how artifactually imprinted protocols are designed and used by actors in everyday work activities, as shown in figure 1. For the purpose of developing Ariadne it was necessary to assume that the identified set of categories of articulation work is complete and definite, but this does not preclude the notation from evolving over time.

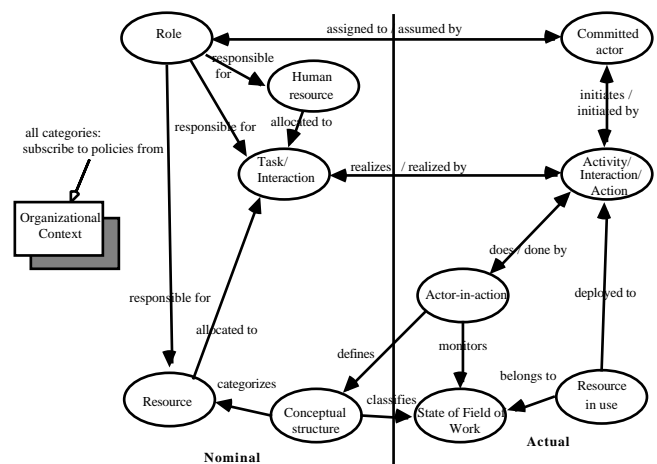


Figure 1. Elemental categories of articulation work model and their cross relations.

It is beyond the scope of this paper to describe these categories. Most of them can be understood intuitively. The category termed ‘conceptual structures’, however, denotes the various constructs needed to express the conceptualizations of the field of work (definitions, classifications, etc.) that the members of the cooperative ensemble have adopted to be able to refer to the multifarious objects and processes of their common field of work in an orderly fashion.

The categories of articulation work are the elements, the basic building blocks, made available by Ariadne. The notation represents the properties of the categories as well as their cross references as attributes of each element: the latter are described in more detail in [5].

As opposed to other notations for constructing protocols within workflow systems, and of CSCW applications in general [1; 4; 7-10; 13; 15], Ariadne does not propose a particular approach to the representation of protocols. Rather, it makes available to the designer a wide spectrum of possibilities. It allows the user to combine the categories required by the protocol construction by means of two general relational structures for representing various relations (causal, hierarchical, instrumental, etc.) among the categories. These structures are: *graphs*, by means of which one can model non-deterministic behavior or relations, and *partial-orders* to represent distributed behavior. This possibility is typically used when a protocol is described as a combination of more elemental protocols, associated to *roles*, interacting among themselves and with the active artifact through the category called *interaction*. Notice that one could avoid any use of the relational structures and exploit the communication capabilities of the categories of the notation. This is typically the case of the activation of the responsibilities and policies within *Tasks*, *Roles* and *Resources* through which protocols can be naturally described in terms of constraints [6; 9].

To support the fundamental role of artifacts in coordination mechanisms, discussed in the previous sections, the notation furthermore contains specific features allowing the designer to define a so-called *active artifact*. An active artifact is defined as a data structure (the artifact) together with communication capabilities that make it ‘active’ in the sense that it actively participates in the coordination of the distributed behavior of components of protocols.

Finally, as illustrated by the scenario, not all of the categories are required for the construction of a particular mechanism. A way of achieving versatility is to allow the designer to select the elements needed to construct the intended mechanism, in particular its protocol. Depending on the characteristics of the protocol, or on individual preferences, the designer may want to adopt a specific approach, for example, an approach focusing on flow of resources among tasks or roles, on the flow of control among tasks, or on communication patterns among roles.

4.2. The primitives of Ariadne

The Ariadne environment is structured in three logical levels (called α , β , and γ , see the central part of figure 2)

that are characterized by the primitives they make available.

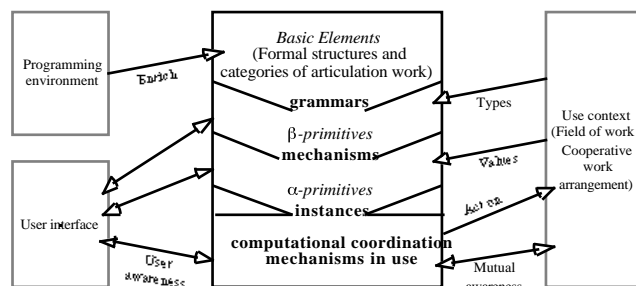


Figure 2. The layered structure of Ariadne and its context

The three levels are not hierarchical, and going from one level to the another level is not identical to a refinement, since the information handled at each level is of a different nature. In fact, at the γ -level, it is possible to define or modify a grammar, thus determining the expressive power of a language for defining a class of computational coordination mechanisms at the α -level. The ‘space of possibility’ within which grammars can be defined at the β -level is determined by the set of categories of articulation work, and by the set of formal structures. At the α -level, it is possible to define or modify a mechanism according to the chosen grammar and make permanent changes to an existing protocol as part of its evolutionary design. At the γ -level, finally, it is possible to instantiate and activate the mechanism in a particular situation and to do so in an incremental fashion. Moreover, the primitives at this level allow for the management of local changes.

While the distinction between the β -level and the α -level of the notation can be recognized in almost all recent CSCW applications, the γ -level is unique. The fundamental rationale for the γ -level is that it provides Ariadne with the versatility required to cope with the immense variety of artifacts, protocols, and interactions. Hence, Ariadne does not impose a preconceived modeling approach. Rather, Ariadne allows designers to adopt a particular combination of the repertory of categories of articulation work to get the desired expressive power.

The arrows on the right-hand side of figure 2 denote the connections of the mechanism to the field of work and the cooperative work arrangement, that is, to its context of use. The definition and specification of a protocol must be related to the particular objects of the given field of work and the work arrangement, as ‘types’, at the β level, and as ‘instances’, at the α level. Thus, Ariadne explicitly requires to specify a clear interface between the computational coordination mechanisms and their context of use. On the other hand, this interface plays the crucial role of a borderline between (a) what is within the realm of articulation work and can be constructed and adapted with Ariadne, and (b) what is in the realm of the objects and processes that are articulated. This distinction, often implicit or underestimated in CSCW literature, has been put

in evidence in the framework of the so called Coordination Languages and Models [3].

5. Linking of coordination mechanisms

The scenario shows that building single, distributed mechanisms is not enough: the next crucial aspect to be considered is how to express the flexible composition of coordination mechanisms in order to support the cooperation across groups of people each of which are 'locally' supported by particular coordination mechanisms. While versatility of modelling is solved by providing high degree of flexibility in the definition of grammars, versatility of composition is solved by the identification of some basic modes in which coordination mechanisms (and, as we shall see, their components) interoperate. In fact, these modes allow for the standardization of the interface to the outside world of each element of the notation, thus enhancing compositionality. A comparative analysis of field study findings led to the identification of three basic modes by means of which (the components of the) mechanisms can be linked by communication.

In *inscription mode* a mechanism provides information about its current state to another mechanism (or, conversely, a mechanism obtains information about the current state of another mechanism). The inscription can be either *compulsory* or *voluntary*, to convey mandatory synchronization request or just awareness.

In *subscription mode* a mechanism makes the behavior of another mechanisms part of its own behavior, for example, by activating another coordination mechanism.

In *prescription mode* a mechanism over-writes the definition of the target mechanism's behavior. In fact, in the prescription mode a given mechanism can change the (nominal) definition of another mechanism, e.g., the definition of its protocol, or its (actual) specification, for example, by enforcing a special state during its execution.

Subscription and inscription modes also apply to the mechanism's components (categories, artifacts, protocols) to express their interactions. For example, a *resource* can be accessed in the subscription mode to activate the policies governing its usage. The compulsory inscription mode typically expresses the reading from and writing to the artifact by the protocols in order to acquire and make visible imperative information. Finally, the voluntary inscription mode is typically used by components and artifacts to create 'awareness' of internal changes to the protocols.

Conversely, prescription mode only applies to *actors* and *roles* within coordination mechanisms since prescription, for security reasons, requires a responsible actor who, at most, can delegate some action to the protocols.

Since *interaction* is the element explicitly devoted to express communication within protocols and across mechanisms, it contains an attribute carrying three values which trigger different behaviors (semantics) according to the above rationale.

6. Using Ariadne

For the sake of illustration, let us suppose that a designer, perhaps one of software designers in our scenario, wants to construct the various coordination mechanisms of the scenario with Ariadne.

The Bug Form and the Binder mechanisms contain protocols characterized by a number of involved *roles* that behave according to a recurrent control flow; moreover, according to the platform integration convention, these protocols must be synchronized in such a way that verification tasks prescribed by all instances of the Bug Report Form mechanism start in parallel. Among the available grammars, the designer decides to use a grammar that is able to express distributed behaviors in terms of clusters (or more formally, partial-orders) of *tasks* and *interactions* associated to *roles*. She can then start to exploit this grammar to describe the behavior of the involved *roles* by using the *interactions* to express the communication between them and with the Bug Report Form artifact.

By using the features of the grammar for describing artifacts, the designer defines a data structure which is suitable for representing the information of the form. In addition, she fills in the section expressing its communication capabilities by taking into account that in the Bug Report Form mechanism the flow of control governing the behavior of the involved *roles* is mediated by the related artifact by the following pattern: a *role* updates the artifact and the latter recognizes this event and triggers the enactment of the behavior of the pertinent *roles*.

The same grammar can be used to construct the Binder mechanism, since it pose similar modeling requirements. However, the Binder artifact deserves some comments since this coordination mechanisms requires a not-trivial integration between its artifact and the protocols of the Bug Report Form mechanisms. In other words, it is not merely a data structure, rather a structure of coordination mechanisms, one for each created bug form. In order to model this situation, the grammar allows the data structure of an artifact to be represented as a list of frames in which the slot containing the identification of a Bug Report Form has as 'value' a link to the related coordination mechanism. Moreover, the Binder Master can organize the list according to different criteria, depending on the view (such as severity, timing) deemed most suitable for the current stage of the project; this is simply represented by the *BinderMaster*'s sending of a message to the *BinderArtifact* in order to activate the suitable function.

Now, let's suppose that our designer has to construct the Project Schedule mechanism. This contains just one *role* (*ScheduleMaster*) whose behavior can be aptly described by the exchange of documents with other *roles* in order to define and modify the plan of activities, and by the interaction with the Project Schedule artifact in order to monitor the execution of each single activity. The designer thinks that the most suitable way to describe the protocol is to represent the flow of suitable documents

among the involved roles. For this the previous grammar is not adequate. Fortunately, someone has already defined the desired grammar (constituted by a non-deterministic graph whose nodes and arcs are labeled `roles` and `informationresources`, respectively). The `informationresources` can be specialized as different types of planning documents (for example, a proposal of a modification or an assignment of work). The interactions between the `Schedule Master role` and the `Project Schedule artifact` exploit another feature of the grammar, namely the fact that `roles` are defined in terms of the `tasks` they are responsible of and the latter have attributed the `activities` needed for their accomplishment: in this case just an `interaction` with the artifact. The latter is constructed in a similar way as the `Bug Report Form artifact`.

In addition, the scenario shows the use of a classification schema, the `Directory`. In a sense, it is similar to the `Binder` mechanism since each software module is represented as a node in the schema. The main difference is that the `Directory artifact` is described as a graph representing the relationships among modules, for instance in a hyper-text-like view. In this case, there is no need for synchronizing behavior (as in the `Binder` and `Bug Report Form` mechanisms); instead, there is a need to provide access to the classification scheme in order to use it as an evolving indexing tool.

To sum up, coordination mechanisms may pose different modeling requirements which, in turn, may require different grammars. It is beyond the scope of illustration to describe these grammars in any detail. The point is that, up to now, at least four different grammars are required in order to implement the scenario. Moreover, the grammars show peculiarities that can hardly be collected in a unified grammar without reducing its usability.

Coming back to the illustration of the use of the notation, the mechanisms have been constructed at this point. Specifically, their links, as described in the scenario, are represented either by `interactions` across mechanisms as in the case of `Binder` and `Bug Report Form` mechanisms, or by the communication capability the language associates to artifacts and to the elements involved in the protocols as `roles` and `informationresources` in the case of the `Directory` mechanism. Thus, the linked coordination mechanisms are available to the users at the `-level` who can activate and manipulate them through the suitable primitives.

Finally let us address the issue of modifications to mechanisms, temporary as well as lasting. Firstly, consider the situation where a `Tester` wants to take the problem directly to the `Designer`, thereby temporarily circumventing the standard diagnosis procedure. This modification can be realized by the application of an `-level` primitive that allows the user to enforce a certain state upon the activated protocol. In this case, the enforcement affects the behavior of the `Tester` who exits the `Bug Report` protocol after the interaction with the `Designer`; it affects the behavior of the `Designer` since it is started by a communication coming from an unexpected source; and finally, it affects the behavior of the `BinderMaster` which now is

started from an intermediate state which is different from its default initial state. A local change like this may, of course, have a global effect, in the sense that the various components of the distributed protocol have to reach a coherent new global state. The coherence of the new 'global' state is the responsibility of the involved actors. In order to reach agreement on the new global state from which to resume the default behavior of the protocol, they may use different communication channels. The point is that the notation allows them to import the effects of this agreement in the running computational mechanism.

Next, consider the situation where the design team wants to introduce a new role to the `Bug Report` protocol, namely the role of a project manager. This modification, a lasting one, can be performed by a `-level` primitive that allows the manipulation of the components constituting a mechanisms (within the space of possibility of the grammar used for its definition). Specifically, a new `role` (`Project Manager`) is added to the `Bug Report` mechanism by an actor authorized to do this. The action of this new `role` is expressed through its capability to communicate with the `Bug Report artifact`. The definition of the `role` is expressed through the specification of the new `task` to be attributed to it (as described above when we sketched the construction of the mechanisms), the definition of the artifact is expressed by inserting the appropriate capability of mediating the communication between the designers, the `Project Manager` and the `Binder Master` (again through event recognition and triggers). This modification will affect the next instantiations of the `Bug Report Form` mechanism.

7. Ariadne and Abaco

In view of the internal complexity of the notation, caused by the number of components together with their flexible composition, we found it appropriate to define an *abstract machine* in terms of which the operational semantics of Ariadne is specified. This abstract machine has a multi-layered architecture of agents (called ABACO) whose communication language (called Interoperability Language) consists of a small set of primitives which express the interoperability modes described previously. In other words, all aspects of the notation (categories, attributes, relations, primitives at the various levels, etc.) are mapped onto (aggregations of) agents interacting by means of the Interoperability Language [5; 14]. This approach guarantees the modularity implied by the requirements of malleability and linkability. However, it also matches the inherent distributedness of cooperative work in that, in this environment, coordination is taken as an emergent property arising from the behavior of each individual component at any level of aggregation.

ABACO has been successfully tested in a demonstrator especially to check its robustness against the basic requirements and a full version is presently being implemented in Java. In this implementation, the behavior of the components constituting the notation is represented inter-

nally as labelled Petri nets in order to provide advanced support for malleability and linkability.

Two major conceptual problems arise from this implementation, however: on one hand, the need of realizing the reflective capabilities imposed by the requirement of malleability in a non-reflective programming environment; on the other hand, the need of managing the implications of local modifications in a architecture so radically distributed. In fact, effective support of the management of propagation of changes requires further theoretical work in distributed systems modelling and checking which takes into account the organizational solutions to this problem devised by actors in charge of the modifications in the real world.

Acknowledgments

The development of the framework outlined in this paper was supported by ESPRIT Basic Research through Action 6225 (COMIC) and by the The Danish Natural Science Research Council. We are especially indebted to Peter Carstensen and Monica Divitini. The paper was written while Kjeld Schmidt was with Risø National Laboratory, Denmark.

References

1. Bogia, Douglas P., and Simon M. Kaplan: 'Flexibility and Control for Dynamic Workflows in the wOrlds Environment,' in N. Comstock et al. (eds.): *COOCS '95. Conference on Organizational Computing Systems, Milpitas, California, August 13-16, 1995*, ACM Press, New York, 1995, pp. 148-159.
2. Carstensen, Peter H., and Carsten Sørensen: 'From the social to the systematic: Mechanisms supporting coordination in design,' *Computer Supported Cooperative Work. The Journal of Collaborative Computing*, vol. 5, no. 4, 1996, pp. 387-413.
3. Ciancarini, Paolo, and Chris Hankin (eds.): *Coordination '96. First International Conference on Coordination Languages and Models*, Springer Verlag, Berlin, 1996. - LNCS1061.
4. De Michelis, Giorgio, and M. Antonietta Grasso: 'Situating conversations within the language/action perspective: The Milan Conversation Model,' in T. Malone (ed.) *CSCW '94. Proceedings of the Conference on Computer-Supported Cooperative Work, Chapel Hill, North Carolina, October 24-26, 1994*, ACM Press, New York, N.Y., 1994, pp. 89-100.
5. Divitini, Monica, Carla Simone, and Kjeld Schmidt: 'ABACO: Coordination mechanisms in a multi-agent perspective,' in *COOP '96. Second International Conference on the Design of Cooperative Systems, Antibes-Juan-les-Pins, France, 12 - 14 June, 1996*, INRIA Sophia Antipolis, France, 1996.
6. Dourish, Paul, Jim Holmes, Allan MacLean, Pernille Marqvardsen, and Alex Zbyslaw: 'Freeflow: Mediating between representation and action in workflow systems,' in M. S. Ackerman (ed.) *CSCW '96. Proceedings of the Conference on Computer-Supported Cooperative Work, Boston, Mass., November 16-20, 1996*, ACM press, New York, N.Y., 1996, pp. 190-198.
7. Ellis, Clarence A.: 'Information Control Nets,' in *Proceedings of the ACM Conference on Simulation, Measurement and Modeling, Boulder, Colorado, August 1979*, 1979.
8. Fuchs, Ludwin, Uta Pankoke-Babatz, and Wolfgang Prinz: 'Supporting cooperative awareness with local event mechanisms: The GroupDesk system,' in H. Marmolin, Y. Sundblad and K. Schmidt (eds.): *ECSCW '95. Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work, 10-14 September 1995, Stockholm, Sweden*, Kluwer Academic Publishers, Dordrecht, 1995, pp. 245-260.
9. Gance, Natalie S., Daniele S. Pagani, and Remo Pareschi: 'Generalized Process Structure Grammars (GPSG) for flexible representations of work,' in M. S. Ackerman (ed.) *CSCW '96. Proceedings of the Conference on Computer-Supported Cooperative Work, Boston, Mass., November 16-20, 1996*, ACM press, New York, N.Y., 1996, pp. 180-189.
10. 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.
11. Norman, Donald A.: 'Cognitive artifacts,' in J. M. Carroll (ed.) *Designing Interaction. Psychology at the Human-Computer Interface*, Cambridge University Press, Cambridge, 1991, pp. 17-38.
12. Schmidt, Kjeld, and Carla Simone: 'Coordination mechanisms: Towards a conceptual foundation of CSCW systems design,' *Computer Supported Cooperative Work. The Journal of Collaborative Computing*, vol. 5, no. 2-3, 1996, pp. 155-200.
13. Shepherd, Allan, Niels Mayer, and Allan Kuchinsky: 'Strudel - An extensible electronic conversation toolkit,' in *CSCW '90. Proceedings of the Conference on Computer-Supported Cooperative Work, Los Angeles, Calif., October 7-10, 1990*, ACM press, New York, N.Y., 1990, pp. 93-104.
14. Simone, Carla, Monica Divitini, Kjeld Schmidt, and Peter Carstensen: 'A multi-agent approach to the design of coordination mechanisms,' in V. Lesser (ed.) *Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, Calif., USA, June 12-14, 1995*, AAAI Press, Menlo Park, Calif., 1995.
15. Swenson, K. D., R. J. Maxwell, T. Matsumoto, B. Saghari, and K. Irwin: 'A business process environment supporting collaborative planning,' *Collaborative Computing*, vol. 1, no. 1, March 1994, pp. 15-24.
16. Waldrop, N. Mitchell: *Complexity. The emerging science at the edge of order and chaos*, Simon & Schuster, 1992, (Paperback ed., Penguin Books, 1994).