

Ariadne

Towards a technology of coordination

Risø-R-943(EN)

Carla Simone, Kjeld Schmidt,
Peter Carstensen, and Monica Divitini

Risø National Laboratory
November 1996

Abstract. This is a report on an attempt to develop a CSCW infrastructure that can support the construction and use of malleable and linkable coordination mechanisms. On the basis of a scenario derived from field studies, the paper outlines the Ariadne framework, describes the notation's main features and illustrates how it can be applied to the scenario. The paper attempts to bridge from sociological investigation to computer science.

Supported by the EU's Esprit Basic Research program (the COMIC project, action no. 6225), and The Danish Natural Science Research Council.

© Carla Simone¹, Kjeld Schmidt², Peter Carstensen², Monica Divitini³

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

²Systems Analysis Dept.,
Risø National Laboratory,
P.O. Box 49,
DK-4000 Roskilde, Denmark.
Email: *{kjeld.schmidt, peter.carstensen}@risoe.dk*

³Dept. of Computer Sciences,
University of Milano,
Via Comelico, 39/41,
I-20135 Milano, Italy.
Email: *divitini@hermes.mc.dsi.unimi.it*

ISBN 87-550-2246-4
ISSN 0106-2840
Grafisk Service, Risø, 1996

Table of contents

1.	A real-world scenario	4
2.	Analysis	7
3.	Computational coordination mechanisms	10
4.	The Ariadne framework	14
4.1.	A lexicon of articulation work.....	15
4.2.	Mutual alignment of coordination mechanisms	18
4.3.	Using the notation	19
5.	Conclusion: where are we now?	23
	References	25

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. The system will, eventually, consist of more than 200,000 lines of code and they have not had previous experience with anything of this scale. Previously they have been working rather independently of each other, from time to time two designers would work together, but most of the time they would have been on their own. 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.

Initially, the team plunged into the project by designing and coding in the way they had been used to — and hit the blank wall of reality with a sickening thud. They were amiss and it was discussed whether the ambitious goals of the project should be abandoned. However, in the dark days of the crisis, the software designers decided to fight it out and they 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 is introduced and used to capture and display the relationships between actors, responsibilities, tasks, and deadlines.

The Project Schedule was embedded in a context in which a particular notion played a crucial role, namely that of the so called platform period. Initially, the 'software platform' was a point in time at which all software designers would stop coding in order to integrate their bits and pieces. For each such platform integration period, one of the designers is appointed Platform Master which implies that he or she will be responsible for collecting information on changes made to the software and for ensuring that the software is tested and corrected before it is released. Before the software is released as a 'platform' for further development, the project schedule is updated with revised plans and tasks for the next three to six weeks.

As an artifact, the Project Schedule is a table with the following categories of information: (1) tasks to be accomplished and a reference to a detailed description of the task; (2) the estimated amount of time per module for each task; (3) the software designer responsible for each module; (4) the platform integration period for which tasks are to be finalized; (5) and the total planned work hours per platform period for each software designer.

¹ Our scenario is based upon field studies at Foss Electric in Denmark. In these field studies different design activities of a major design effort, the 'S4000 project', were investigated. The field study findings have been reported elsewhere (e.g., Carstensen et al., 1995a; Carstensen et al., 1995b).

The Directory. The directory structure was developed and used to support the software integration. The designers had to place their personally tested software in a pre-specified directory structure before the deadlines of the integration periods.

The directory structure of the software module repository provides a classification scheme for handling software modules. In addition to enabling distributed storage and retrieval of modules, the directory structure provides a common standardized conceptualization of essential aspects of the field of work, namely the software architecture, i.e., it settles the basic structure of the software which all involved actors have to relate to in their work. 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.

As an artifact, the Bug Report Form is a paper form. Its agreed-to protocol dictates that when a new bug is detected by anyone involved in testing the software, e.g., software designers, other designers, quality assurance staff, and marketing people, a new bug report is initiated and filled-in. The originator of the bug report also provides a preliminary description and diagnosis of the problem. The ‘spec-team’, i.e., three software designers responsible for diagnosing and classifying all reported bugs, then determines the presumably culpable module and, by implication, the designer responsible for that module and therefore for correcting the bug; 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 (handling ‘his’ or ‘her’ bugs) 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 following categories: (1) non-corrected ‘catastrophes’, (2) non-corrected ‘semi-serious’ problems, (3) non-corrected ‘cosmetic’ problems, (4) postponed, (5) rejected, (6) corrected but not yet tested, and (7) corrected bugs. The forms collected in the binder are successively re-classified and re-filed according to decisions made by the spec-team, messages concerning specific bugs from the designers, results from the verification of the reported corrections from the platform integration period, etc.

From time to time, however, the software designers will experience situations where a protocol they have devised and adopted does not seem to provide adequate coordination and where they therefore modify the protocol accordingly. These modifications may have different scope. On one hand, changes 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. The motivation is obviously that the tester happens to know who is responsible and he

expects it to be very easy to solve the problem. On the other hand, the actors may want to change the protocol for good. The software designers may, for example, want to prioritize how effort is spent in debugging the software. To achieve this, they introduce the role of a project manager in the protocol and the Bug Report protocol is changed so that a correction task requires authorization before it is initiated. Likewise, the directory structure is adjusted continually according to the changing architecture of the total software system and to the distribution of responsibilities for the modules among software designers.

It is obvious from our description that the different protocols intersect and must be aligned somehow by the actors. When using the bug report form, for instance, participants will consult the project schedule to see the name of the designer that will be responsible, as 'platform master', for verifying a presumably corrected bug; this is implicitly indicated in the bug report form by the number of the platform period. A platform period number in the bug report form also implicitly indicates a deadline for the correction task to be finished. This deadline is not explicitly stated in the bug report but is again inferred from the reference to the platform period or, in other words, through a 'subscription' to the project schedule spreadsheet where the deadline is stated explicitly.

The case also demonstrates a more active form of interaction between protocols, namely in the form of one protocol inciting the execution of another protocol when a certain condition occurs. For example, when a reported bug is accepted as a bug, a new task is announced and entered in the project schedule, i.e., the change in the state of one mechanism (the bug report form) triggers operations on data structures of another mechanism (the project schedule). Similarly, when a bug is reported to have been corrected, yet another task is announced, namely the task of verifying the correction. However, this task is not initialized until the 'platform master,' who will be responsible for the verification, has been appointed and until the next integration period has started. This starting point is specified in the project schedule.

2. Analysis

The assemblage of protocols described in this scenario could, of course, be modelled as an integrated workflow. The point of our analysis, however, is that what we have here is more than a workflow. There is a specific and crucial relationship between protocol and artifact.

On one hand, the protocol is objectified in the artifact. The artifact thereby gives a certain permanence to the protocol. That is, it conveys the stipulations of the protocol in a situation-independent manner. Moreover, the general stipulations of the protocol are, to some extent, conveyed by the ‘material format’ of the artifact, e.g., the tabular form of the Project Schedule, the selection of blank fields of the bug report form, the hierarchy of the directory, and the separators of the binder. As observed by Goody, the spatio-graphic format of an artifact can stipulate behavior by reminding an actor of items to do and directing attention to missing items: ‘The table abhors a vacuum’ (Goody, 1987, p. 276). This is, again, eloquently illustrated in the case of the bug form where the bug report form provides a set of fields which match crucial points of the bug handling protocol and which are to be filled in by the different actors in the course of the bug’s life

On the other hand, the artifact provides a representation of the state of the execution of the protocol and thereby serves as an *intermediary* between actors that 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. Firstly, the form is transferred from one actor to another and this change of location transfers to the particular actor the specific responsibility of taking such actions on this particular bug that are appropriate according to the agreed-to protocol. Secondly, 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,’ a space with a particular structure that reflects salient features of the protocol. The artifact supporting the directory plays a similar role, albeit with differences. The directory protocol is a classification scheme, that is, it does not stipulate what is to be done in a certain situation but captures certain agreed-to categories and their relationships, in this case part-whole relationships, and provides an index to the location of items of these categories. It thus assists the actors, in their distributed activities, in handling myriad items in an orderly fashion.

In each instance, the artifact plays a crucial role for the protocol by giving the protocol permanence, by objectifying salient concerns of the protocol, by mediating the state of the protocol among actors, etc. In other words, the arti-

fact and the protocol work ‘hand in glove’. We call such a protocol-cum-artifact that is used for coordination purposes a coordination mechanism.

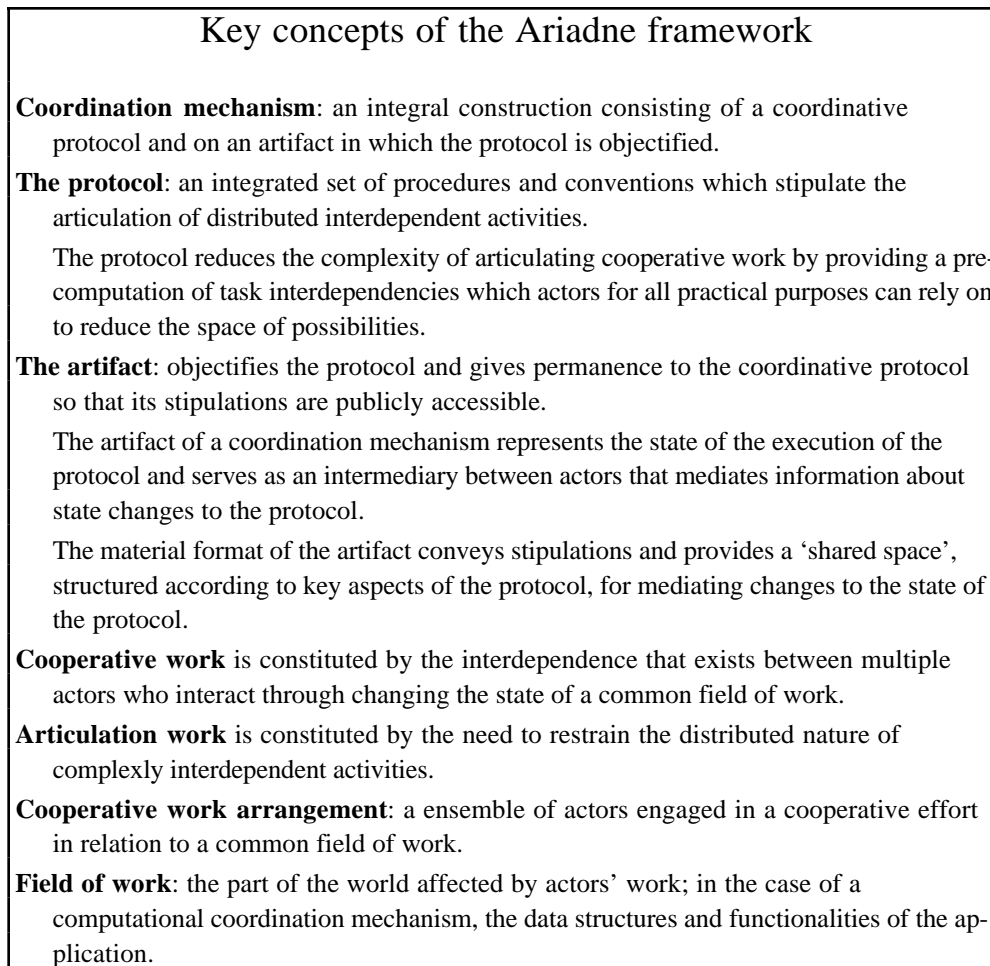


Figure 1. Key concepts of the Ariadne framework.

Furthermore, each of these protocol-cum-artifacts or coordination mechanisms is devised to serve specific purposes in the setting. They address a very narrow set of coordinative activities. However, as observed above, these mechanisms intersect at various points and therefore need to be aligned by the actors in the course of their work but they are only loosely coupled. Because of that, each coordination mechanism can be constructed relatively independently of the others. Thus, in designing a mechanism for a specific purpose, one does not need to have an overview of the totality of work processes in the setting at large. In fact, because such mechanisms can be designed and introduced to serve limited purposes, the array of coordination mechanisms of the work arrangement at large can be designed and maintained in a distributed and bottom-up manner.

From this case (and, indeed, other cases) we may derive an approach to the design of workflow systems, in line with Davenport’s suggestions

(Davenport, 1993), which may result in workflow systems that are far less brittle in the face of the vicissitudes of contemporary business environments than workflow systems often seem to be.

3. Computational coordination mechanisms

Coordination mechanisms based on paper artifacts (e.g., forms, catalogues, time tables) have been around for ages and they are used on a massive scale in modern work settings. While mundane and unassuming, they have crucial affordances: (a) the artifact can represent and convey stipulations among actors in a permanent and publicly accessible form; (b) the protocol and the artifact can be defined and specified by the actors themselves — operators, clerks, managers, auditors, etc. — by means of the ordinary skills of their professions; (c) actors have total control of the interpretation and execution of the protocol and can, under conditions of social accountability, modify or deviate from the protocol; (d) the artifact can dynamically represent state changes to the protocol and mediate these among actors, and (e) multiple coordination mechanisms can be aligned, seamlessly and smoothly, by actors.

Nonetheless, such mechanisms have serious inherent limitations: (a) state changes to the protocol are conveyed by paper and similar unwieldy artifacts and the speed and pattern of propagation of changes to the state of the protocol are thus severely limited; (b) the protocol is only immediately visible to actors to the extent that the protocol is mapped onto the material format of the artifact that serves as an intermediate; (c) modifications to the protocol only take effect when or if actors become aware of them through other channels; (d) local and temporary deviations from the protocol may not be visible to other actors and may cause confusion; (e) maintaining a conventional, paper-based coordination mechanism involves a plethora of mind-numbing operations; (f) it involves massive housekeeping efforts and it may thus be practically impossible for actors to obtain an overview of the state of the protocol, and (g) the seamless and smooth alignment of multiple coordination mechanisms is only feasible insofar as the same actors are involved with the coordination mechanisms in question.

These limitations with conventional coordination mechanisms become increasingly problematic as modern industrial, service, and administrative organizations need to be able to operate in a radically flexible and adaptive and yet highly coordinated fashion. Now, given the infinite versatility of computer systems, it is our contention that computer-based coordination mechanisms can provide a degree of visibility and flexibility to coordination mechanisms that was unthinkable with previous technologies, typically based on inscriptions on paper or cardboard. This opens up new prospects for moving the boundary of allocation of functionality between human and artifact with respect to articulation work so that much of the drudgery of articulation work (boring operations that have so far relied on human effort and vigilance) can be delegated to the artifact, but also, and more importantly, so that cooperative ensembles can articulate their distributed activities more effectively and with a higher degree of flexibility and so that they can tackle an

even higher degree of complexity in the articulation of their distributed activities!

In fact, since the early days of CSCW, it has been a major research issue to understand how computer systems can be instrumental in reducing the complexity of coordinating cooperative activities, individually conducted and yet interdependent. The initial results of this line of research were not encouraging, however, in that coordination facilities were experienced as excessively rigid, either because the underlying protocol was not accessible and could not be modified (e.g., The Coordinator, cf. Flores et al., 1988), or because the facilities for changing the protocol did not support actors themselves in modifying the protocol (e.g., DOMINO, cf. Kreifelts et al., 1991). In response to these initial experiences, a number of research projects have attempted to make coordination facilities flexible to actors, e.g., Egret (Johnson, 1992), ConversationBuilder (Kaplan et al., 1992; Bogia et al., 1993; Bogia et al., 1996), OVAL (Malone et al., 1992), and Regatta (Swenson et al., 1994).

While closely related to these and other CSCW research efforts, our research has taken a somewhat different approach in that we have aimed at developing a computational notation which, on one hand, is sufficiently general to facilitate the construction of computer-based coordination mechanisms for any cooperative work arrangement and which, on the other hand, supports the cooperating actors themselves in constructing mechanisms which are both malleable and linkable.

We define 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.

Notice that, as far as the computational protocol is concerned, only *aspects* of the protocol are incorporated. A protocol only has its (more or less) certain and agreed-to meaning within a certain social context, and it is inescapably under-specified (Suchman, 1982, p. 411). A computational coordination mechanism is invariably embedded within a social context of conventions and tacit assumptions which competent members take for granted but which are not amenable to incorporation in the computational protocol. Hence, a computational protocol cannot simply replace members' more or less tacit conventions and social competencies. This means that the precise allocation of functionality between human actors and computational coordination mechanism is a non-trivial analysis and design task. Furthermore, for every computational coordination mechanism, there will exist facets of the protocol which are not incorporated in the computational protocol. That is, in the construction of a computational coordination mechanism, the protocol is split into a computational protocol and a residual non-computational or 'social' protocol. A com-

putational coordination mechanism is not an immaculate reincarnation of a coordination mechanism.

On the basis of the above scenario — and, in fact, a range of focused field studies — we have found that a 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, when leaving the realm of the *nominal* definition of the protocol for its actual instantiation and execution, actors must be able to control the execution of 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.

Similarly, 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.

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.

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 its organizational context. Similarly, since cooperative work and its articulation is always done in the context of a particular work domain and with reference to specific objects and processes of the common field of work, actors must be able to link the definition and specification of the mechanism to entities in the field of work as represented by the data structures and functionalities of the software applications in which it is embedded.

However, our scenario eloquently illustrates the problems one faces when undertaking to construct computational coordination mechanisms that can satisfy these requirements. In constructing these mechanisms, we must be able to construct artifacts and protocols that can interact and interact in different ways, just like mechanisms can be linked and linked differently. Also, the artifacts are of different nature and pose quite different modeling problems, as do the various protocols. In short, to be able to construct computational co-

ordination mechanisms that meet these requirements, we need a highly versatile tool. The Ariadne framework is conceived exactly for that purpose.²

² In Greek mythology, Ariadne was the daughter of Minos, the king of Crete. She fell in love with Theseus and helped him slay the Minotaur, the monster of the Labyrinth. She did that by giving Theseus a thread that would help him to find his way in and out of the Labyrinth.

4. The Ariadne framework

Ariadne is meant to provide a general infrastructure for CSCW applications, specifically dedicated to the construction, maintenance, execution, and mutual alignment of computational coordination mechanisms embedded within the applications.

Ariadne is, basically, a notation, i.e., a system of symbols and rules for their application³. It is structured in three logical levels that are called α , β , and γ (see the central part of figure 2) and can be described as follows.

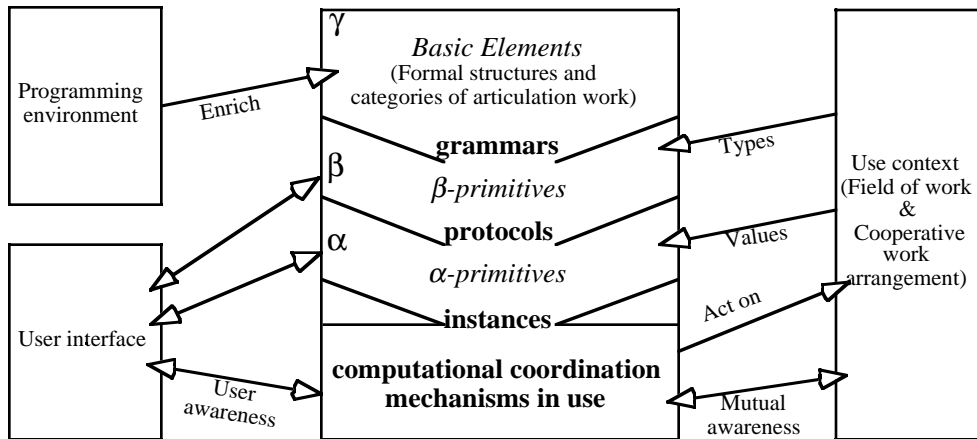


Figure 2. The layered structure of Ariadne and its context

The three levels are not hierarchical, and going from one level to the lower level is not identical to a refinement, since the information handled at each level is of a different nature. Rather, each level defines the ‘space of possibility’ for the lower one: the γ -level provides the grammars that can be applied at the β -level to define protocols, whereas the protocols defined at the β -level can be specified at the α -level. More specifically:

At the γ -level, it is possible to define or modify a grammar which can then be used to construct a variety of protocols at the β -level. Building a grammar means determining the expressive power of a language for defining a class of computational coordination mechanisms. The ‘space of possibility’ within which grammars can be defined at the γ -level is determined by a set of categories of articulation work underlying Ariadne at any point in time, and by the available set of formal structures for representing various relations (causal, hierarchical, instrumental, etc.) among the categories. These categories are described in section 4.1 below.

At the β -level, it is possible to define or modify the protocol itself according to the chosen grammar and make permanent changes to an existing proto-

³ According to Wittgenstein, ‘Grammar does not tell us how language must be constructed in order to fulfill its purpose, in order to have such-and-such an effect on human beings. It only describes and in no way explains the use of signs.’ (Wittgenstein, 1958, § 496)

col as part of its evolutionary design. At the α -level, finally, it is possible to instantiate and activate the protocol 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. The aim is to overcome a certain limitation of workflow systems, and of CSCW applications in general, namely that they impose a particular modelling approach (Ellis, 1979; Cook, 1980; Shepherd et al., 1990; Medina-Mora et al., 1992; Swenson et al., 1994). Accordingly, Ariadne does not impose a preconceived modeling approach. Rather, Ariadne allows designers to adopt a particular interpretation of the repertory of categories of articulation work.

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 a clear interface (although very elementary at this stage) between the computational coordination mechanisms and their context of use.

The different levels of Ariadne will typically be accessed by users with different skills. The α - and β -levels are typically needed by end-users who, as part of their everyday work activities, use and design coordination mechanisms. At these levels the use of the notation will merely require the ability of selecting and combining predefined items according to the rules of a relevant grammar. The γ -level, on the other hand, is typically the realm of the ‘application designer’ or, in our framework, actors who define grammars needed by a special community of end-users for defining coordination mechanisms.

4.1. A lexicon of articulation work

As for the modeling of protocols, the required versatility is achieved through the identification of categories that are flexibly composed via a set of predefined relational structures.

The crucial point in developing a notation for constructing computational coordination mechanisms is to determine a repertory of elemental categories, at the semantic levels of articulation work, by means of which coordinative protocols can be expressed. Taking Anselm Strauss’ quite informal lexicon of articulation work (who, what, where, when, how, how soon, for how long, etc. (Strauss, 1985)) as our baseline, a set of elemental categories was de-

rived from the studies of how artifactually imprinted protocols are designed and used by actors in everyday work activities, as shown in the table of figure 3. These categories represent the minimal set of categories required to express the protocols examined in these field studies.

Nominal (for the definition of protocols)		Actual (for the specification of protocols)	
Elemental categories of articulation work	Predicates	Elemental categories of articulation work	Predicates
<i>Articulation work with respect to the cooperative work arrangement</i>			
Role	assign to [Committed actor]; responsible for [Task, Resource]	Committed-actor	assume, accept, reject [Role]; initiate [Activity];
Task	point out, express; divide, relate; allocate, volunteer; accept, reject; order, countermand; accomplish, assess; approve, disapprove; realized by [Activity]; to be aligned with [Task]	Activity/Action	[Committed actor] initiate; [Actor-in-action] undertake, do, accomplish; realize [Task]; [Actor-in-action] makes publicly perceptible, monitors, is aware of, explains, questions; aligned with [Activity]
Human resource	locate, allocate, reserve;	Actor-in-action	initiates [Activity]; does [Activity];
<i>Articulation work with respect to the field of work</i>			
Conceptual structures	categorize: define, relate, exemplify relations between categories pertaining to [Field of Work];	State of field of work	classify aspect of [State of field of work]; monitor, direct attention to, make sense of, act on aspect of [State of field of work];
Informational resource	locate, obtain access to, block access to;	Informational resources-in-use	show, hide content of; publicize, conceal existence of;
Material resource	locate, procure; allocate, reserve to [Task];	Material resources-in-use	deploy, consume; transform;
Technical resource	locate, procure; allocate, reserve to [Task];	Technical resources-in-use	deploy; use;
Infrastructural resource	reserve;	Infrastructural resources-in-use	use;

Figure 3. Elemental categories of articulation work model and their predicates.

For the purpose of developing Ariadne it was necessary to assume that the identified set of elemental categories of articulation work is complete and definite, but this does not preclude the lexicon from evolving over time. In fact, this set of categories has been derived empirically through an iterative process of induction, and the repertory is undoubtedly neither complete and nor definite. Moreover, this lexicon is not intended to be a comprehensive terminology for any kind of empirical investigations of cooperative work, since it was derived solely for the purpose of expressing protocols with a view to constructing computational mechanisms of interaction.

In figure 3 the elemental categories and their predicates (properties and cross references) are ordered along two dimensions: vertically, categories of articulation work with respect to the *cooperative work arrangement* versus *the field of work*; horizontally, categories of *nominal* versus *actual* articulation work. Specifically, the category termed ‘conceptual structures’ 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 properties of the categories as well as their cross references are represented by attributes of each element. The elements with their attributes are described in more detail in (Simone et al., 1995a).

To support the fundamental role of artifacts in coordination mechanisms, namely to objectify the protocol and make its stipulations publicly accessible, the notation furthermore contains specific features which by default are incorporated in each grammar, thereby 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 behaviour 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 first 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.

Moreover, when needed, a protocol can be given a distributed character. In this case, the component parts of the protocols interact among themselves and with the active artifact to describe the distributedness of the protocol. In order to achieve this it is appropriate to describe the protocol as a combination of more

elemental protocols, or *proctors*. If distributedness is not a primary concern, the protocol coincides with a single proctor, it is ‘monolithic’.

As opposed to other notations for constructing protocols (e.g., Regatta, Action Workflow, or the Milano System), Ariadne does not propose a particular approach to the representation of protocols. Rather, it makes available to the designer a wide spectrum of possibilities, just by combining the elements required by the protocol construction by means of two general relational structures. These are: *graphs*, by means of which one can model non-deterministic behaviour or relations, and *partial-orders* to represent concurrent or distributed behaviour. Notice that one could avoid any use of the relational structures and exploit the communication capabilities of the elements of the notation (typically, the activation of the responsibilities and policies within `Tasks`, `Roles` and `Resources`) to specify protocols characterized by ‘light’ coordination needs.

4.2. Mutual alignment of coordination mechanisms

The scenario shows that building single mechanisms is not enough: the next crucial aspect to be considered is how to express the composition of specific coordination mechanisms in order to support cooperation across groups of people ‘locally’ supported by each specific coordination mechanism. This is the point where versatility in relation to communication requirements plays its role. While versatility of protocol modelling is solved by providing high degree of flexibility in the definition of grammars, versatility of communication is solved by the identification of some basic modes in which coordination mechanisms (and, as we shall see, their components) interoperate. A comparative analysis of field study findings led to the identification of three basic modes by means of which 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, that is, 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 (elements, artifacts, proctors) to express their interactions. For example, a `resource` can be accessed in the subscription mode to activate the poli-

cies 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 among proctors and across mechanisms, it contains an attribute carrying three values which trigger different behaviours according to the above rationale.

We are not claiming that identified modes are fully original or exhaustive: they can be related, for example, to well-known features of programming languages (e.g., call by reference, co-routines, reflective features). The point is that these three modes express the capabilities provided by the more advanced programming languages at the semantic levels of articulation work and that the combined use of them is able to represent all the needs of linking mechanisms as identified in our studies (up to now).

4.3. Using the notation

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 behaviours 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 behaviour 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 behaviour 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 behaviour (proctors) 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 (`Schedule Master`) whose behaviour can be aptly described by the exchange of documents with other roles (namely, `Spec-Team`, `Designer` and `Project Manager`) 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 `information resources`, respectively). The `information resources` 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.

Finally, in the scenario two classification schemes are present: the first one classifies Bugs, the second one (the Directory) defines the structure of the software modules. Of course, none of the two grammars used previously are adequate for constructing these. Moreover, the two cases pose different modeling requirements since, as it is quite evident, the complexity of the two situations is very different. In fact, in the first case, the Bug Classification artifact contains just the name and description of the various bug classes: some `role` is responsible for its `maintenance`. The `maintenance task` is defined in terms of `activities` suitable for governing the collection of complaints and proposed changes to the classification and for agreeing on a change.

The second classification schema, the Directory, is similar to the Binder mechanism in that each software module is represented 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 hypertext-like view. In this case, there is no need for synchronizing behaviour (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 *information resources* in the case of the Directory mechanism. Thus, the linked coordination mechanisms are available to the users at the α -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 *Spec-team*. This modification can be realized by the application of an α -level primitive that allows the user to enforce a certain state upon the activated protocol. In this case, the enforcement affects the behaviour 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 behaviour of the *Binder Master* 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 behaviour 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 β -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` (`ProjectManager`) 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` (let's call it `Authorization`) 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 `Spec-team`, the `ProjectManager` and the `BinderMaster` (again through event recognition and triggers). This modification will affect the next instantiations of the Bug Report Form mechanism.

5. Conclusion: where are we now?

The development of Ariadne as a general notation for constructing coordination mechanisms has involved a range of research activities.

Firstly, we needed to understand how cooperating actors devise and use coordinative constructs such as coordinative protocols and how such constructs are supported by artifacts. A series of focused, in-depth field studies of coordinative practices in real-world cooperative settings have therefore been undertaken with the specific objective of investigating how the distributed activities of cooperative work arrangements are articulated and, in particular, how prescribed procedures and artifacts are devised, appropriated, and used for these purposes. As part of the analysis of the field studies, a prototype of CSCW application for supporting the coordination of software testing was developed.

On the basis of these activities we have developed a general conceptual framework which provides a set of categories and relations which, in turn, form the basis of Ariadne. The whole process has been highly iterative, of course, in that the development of Ariadne has raised questions to the underlying conceptual framework which, again, has generated issues to be investigated further in the field studies. On the other hand, the findings from the field studies have been used to put the conceptual framework and the notation to test; as a result the framework and the notation have been modified and refined repeatedly.

In the initial development of Ariadne, a decision was made to postpone the implementation and concentrate on developing a formal specification of its elements and on evaluating it against the requirements and scenarios derived from field studies. This strategy was adopted, consciously and explicitly, in order to avoid having the notation influenced, in an implicit and uncontrollable manner, by the inevitable limitations of currently available implementation platforms. Presently, the realization of Ariadne is under development but far from being concluded, for many reasons. First of all, as anticipated, the choice of the elements and of the relational structures is not considered as definitive.

Secondly, the internal complexity of the notation, caused by the number of components together with their flexible composition, makes it necessary the construction of an *abstract machine* in the terms of which to define the operational semantics of Ariadne. This abstract machine is constituted of a multi-layered architecture of agents (called ABACO) whose communication language (called Interoperability Language) contains a small set of primitives realizing the linking modes described previously. In other words, all aspects of the notation (components, relations, primitives at the various levels, etc.) are mapped on (aggregations of) agents interacting by means of the Interoperability Language (Divitini et al., 1995; Simone et al., 1995b). This formal

specification has been tested in a demonstrator especially to check the robustness of ABACO against the basic requirements of malleability and linkability.

Thirdly, the requirements of malleability and linkability are presently only partially satisfied. More sophisticated primitives are needed together with their mapping in ABACO. This enrichment should involve capabilities of supporting the modification process as well as the management of the various versions (e.g., Bogia and Kaplan, 1995). However, the ABACO machine allows us to reason about the implementation of Ariadne at an abstraction level commensurable with the semantic level of articulation work at which Ariadne is defined and at which malleability and linkability make sense to the users. The same holds for any conceivable extension of expressive power and functionality that must be added in such a way that the current 'orthogonality' of the component of ABACO is preserved.

References

- Bogia, Douglas P., and Simon M. Kaplan: 'Flexibility and Control for Dynamic Workflows in the wOrlds Environment,' in Nora 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.
- Bogia, Douglas P., William J. Tolone, Celsina Bignoli, and Simon M. Kaplan: 'Issues in the Design of Collaborative Systems: Lessons from ConversationBuilder,' in Dan Shapiro, Michael Tauber, and Roland Traünmüller (eds.): *The Design of Computer Supported Cooperative Work and Groupware Systems*, North Holland, Amsterdam, 1996, pp. 401-422.
- Bogia, Douglas P., William J. Tolone, Simon M. Kaplan, and Eric de la Tribouille: 'Supporting Dynamic Interdependencies among Collaborative Activities,' in Simon Kaplan (ed.): *COOCS '93. Conference on Organizational Computing Systems, Milpitas, California, November 1-4, 1993*, ACM Press, New York, 1993, pp. 108-118.
- 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, 1995a, pp. 314-333.
- Carstensen, Peter H., Carsten Sørensen, and Tuomo Tuikka: 'Let's Talk About Bugs!,' *Scandinavian Journal of Information Systems*, vol. 7, no. 1, 1995b, pp. 33-54.
- Cook, Carolyn L.: 'Streamlining office procedures - An analysis using the information control net model,' in *National Computer Conference, 1980*, 1980, pp. 555-565.
- Davenport, Thomas H.: *Process Innovation: Reengineering Work through Information Technology*, Harvard Business School Press, Boston, Mass., 1993.
- 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].
- Ellis, Clarence A.: 'Information Control Nets,' in *Proceedings of the ACM Conference on Simulation, Measurement and Modeling, Boulder, Colorado, August 1979*, 1979.
- 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.
- Goody, Jack: *The Interface Between the Written and the Oral*, Cambridge University Press, Cambridge, 1987.
- Johnson, Philip: 'Supporting Exploratory CSCW with the EGRET Framework,' in Jon Turner and Robert 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.
- Kaplan, Simon M., William J. Tolone, Douglas P. Bogia, and Celsina Bignoli: 'Flexible, Active Support for Collaborative Work with Conversation Builder,' in Jon Turner and Robert 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.

- Kreifelts, Thomas, Elke Hinrichs, Karl-Heinz Klein, Peter Seuffert, and Gerd Woetzel: 'Experiences with the DOMINO Office Procedure System,' in Liam Bannon, Mike Robinson, and Kjeld Schmidt (eds.): *ECSCW '91. Proceedings of the Second European Conference on Computer-Supported Cooperative Work, 24-27 September 1991*, Kluwer Academic Publishers, Amsterdam, 1991, pp. 117-130.
- Malone, Thomas W., Hum-Yew Lai, and Christopher Fry: 'Experiments with Oval: A Radically Tailorable Tool for Cooperative Work,' in Jon Turner and Robert 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.
- Medina-Mora, Raul, Terry Winograd, Rodrigo Flores, and Fernando Flores: 'The Action Workflow Approach to Workflow Management Technology,' in Jon Turner and Robert 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.
- Shepherd, Allan, Niels Mayer, and Allan Kuchinsky: 'Strudel - An Extensible Electronic Conversation Toolkit,' in *CSCW 90, Los Angeles, CA, October 7-10 1990*, New York, N.Y., ACM press, 1990, pp. 93-104.
- Simone, Carla, Monica Divitini, and Kjeld Schmidt: 'A notation for malleable and interoperable coordination mechanisms for CSCW systems,' in Nora Comstock et al. (eds.): *COOCS '95. Conference on Organizational Computing Systems, Milpitas, Calif., August 13-16, 1995*, ACM Press, New York, 1995a, pp. 44-54.
- Simone, Carla, Monica Divitini, Kjeld Schmidt, and Peter Carstensen: 'A Multi-Agent Approach to the Design of Coordination Mechanisms,' in Victor Lesser (ed.): *Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, Calif., June 12-14, 1995*, AAAI Press, Menlo Park, Calif., 1995b.
- Strauss, Anselm: 'Work and the Division of Labor,' *The Sociological Quarterly*, vol. 26, no. 1, 1985, pp. 1-19.
- Suchman, Lucy A.: 'Systematics of Office Work. Office Studies for Knowledge-Based Systems, Digest,' in *Office Automation Conference, San Francisco, April 5-7, 1982*, 1982, pp. 409-412.
- 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.
- Wittgenstein, Ludwig: *Philosophical Investigations*, (Manuscript 1945-49), Translated by G. E. M. Anscombe, Basil Blackwell Publishers, Oxford, 1958.

 Title and author(s)

Ariadne: Towards a technology of coordination

Carla Simone, Kjeld Schmidt, Peter Carstensen, and Monica Divitini

 ISBN
87-550-2246-4

 ISSN
0106-2840

 Dept. or group
Systems Analysis
Dept.

 Date
November 1996

 Group's own reg.
number(s)

 Project/contract no.

EU Esprit Basic
Research action no.
6225, and The
Danish Natural
Science Research
Council.

 Pages

Tables

Illustrations

References

27

2

1

23

 Abstract

This is a report on an attempt to develop a CSCW infrastructure that can support the construction and use of malleable and linkable coordination mechanisms. On the basis of a scenario derived from field studies, the paper outlines the Ariadne framework, describes the notation's main features and illustrates how it can be applied to the scenario. The paper attempts to bridge from sociological investigation to computer science.

 Descriptors INIS/EDB

 Available on request from

Information Service Department, Risø National Laboratory,
(Afdelingen for Informationservice, Forskningscenter Risø),
P. O. Box 49,
DK-4000 Roskilde, Denmark
Telephone (+45) 4677 4677, ext. 4004/4005
Telex 43 116. Telefax (+45) 4675 5627.