

# Computer-Supported Deliberations for Distributed Teams

Jacques Lonchamp, Fabrice Muller

LORIA, BP 254, 54500 Vandoeuvre-lès-Nancy, France  
{jloncham, fmuller}@loria.fr

**Abstract.** In our terminology, a “deliberation” is a distributed collaborative process, more or less spontaneous, structured, and complex. This process can include both individual and collective activities, synchronous and asynchronous phases. But in every case, its core part is the negotiation of some result through argumentation. This paper describes a comprehensive flexible generic support for distributed deliberations: motivations, design choices, current Java prototype, and usage through a collaborative document inspection example.

## 1 Introduction

In our terminology, a “deliberation” is a distributed collaborative process, more or less spontaneous, structured, and complex. It can include both individual and collective activities, synchronous and asynchronous phases. But in every case, *at the heart of a deliberation is the negotiation of some result by exchanging arguments*. The collective elaboration of some artifact (e.g. a requirements document) and the collective assessment of an individually proposed artifact (e.g. a code inspection) are classical examples of deliberation types. In classical organizations, such processes heavily rely on face to face meetings. In new distributed and temporary forms of organizations (virtual teams, virtual enterprises, multinational research projects), there is a strong need for a specialized computer support for such processes. We claim that no comprehensive and flexible computer support is currently available. Most current tools emphasize a single aspect of cooperative work: communication (e-mail, chat, bulletin boards, Internet telephony, video conference, ...), coordination (workflow management systems), or collaborative production (shared editors, shared document repositories, ...). Deliberations require a comprehensive support for all these aspects, and it is not easy to build a satisfying support with fragmented tools. Integrated environments could be better, but most of them are either restricted in their scope (e.g. synchronous environments based on the room metaphor, workflow environments with a loose integration of groupware), or so complex, that they appear not suitable for temporary and rather light weight processes. Our research project aims at providing *a comprehensive and flexible generic support for distributed deliberations*.

The rest of the paper is organized as follows. Section 2 introduces the requirements. Section 3 summarizes the main design choices. Section 4 describes DOTS

(‘Decision Oriented Task Support’), our current Java prototype, its architecture, and shows its usage through a collaborative document inspection example.

## 2 The Requirements

There exists a wide range of deliberation types. For instance, a brainstorming process can include a single synchronous session, where participants propose and discuss ideas. Another brainstorming process, can include individual parallel activities for ideas proposal (either private, public, or semi public), followed by a synchronous phase for collective discussion of the individual proposals. Our aim is to *build a generic system which can support this wide range of processes*. Deliberation models should be parameters of the generic system (R1). The system should support structured processes, with both individual and collective activities, and their coordination (R2). The system should support both synchronous and asynchronous phases (R3). In many cases, *the process organization is defined opportunistically by the participants when they participate it*. For instance, in a multi-disciplinary design solution assessment in the aeronautical field [8], the participants can choose first an analytical (criteria based) assessment of the initial solution. Then, if no convergence occurs, and if other solutions are proposed, they can continue with comparative assessments of the alternative solutions. They can also choose analogical assessments with past solutions. The process model should define different strategies for reaching the final goal, letting the participants (all of them or only a ‘process manager’) make the tactical choice of the effective process at execution time (R4).

At the heart of a deliberation is the negotiation of some result by exchanging arguments. The system should support the core argumentation and decision activity (R5). Argumentation can take different forms: in some design activities, arguments are related to precise criteria and can be weighted quite precisely. In other applications arguments are less formal and no weighting is possible. The system should provide a flexible argumentation and decision scheme (R6). In many deliberations, the importance of an argument is related to the person that emitted it and to the role of this person in the organization (‘arguments of authority’). The system should support actor and role identification (R7).

The system should support distributed work, because temporary alliances of partners are the typical target of the system (R8). Due to the temporary nature of these networks, the system should be easy to install and use (R9). In distributed teams, interaction can occur either indirectly, by sharing documents, or directly, through dedicated synchronous or asynchronous communication channels. Direct communication can have different levels of formality. The system should support all kinds of communication: direct/indirect, synchronous/asynchronous, formal/informal (R10). It should manage a range of possible artifacts (R11). For instance, problems, solutions, constraints, in a solution assessment deliberation process. In distributed teams, participants need both guidance and awareness facilities, for answering the classical who? when? where? what? how? questions. The system should provide those facilities, together with less usual argumentation and decision making assistance (R12).

## 3 The Main Design Choices

### 3.1 A Meta Model for Describing Deliberation Models

Our deliberation support system is a generic process-centred system (R1). For building a dedicated environment, a deliberation model describing a set of types and their relations is written with a specific modelling language. This model is instantiated, partly with an instantiation tool, for instances that are necessary for starting the task, and partly during the model execution. Our modelling language is defined by a meta model, extending a classical workflow process meta model with a decision-oriented view [4]: we call our approach “*fine grain decision-oriented task modelling*”. The meta model includes:

- *a process view*, with task and phase types; the instantiation builds a network of phase instances, with precedence relationships;
- *a product view* (R11), with artefact types, specializing text, list, graph, or image generic types, and fine grained component types, specializing list element, graph node, or graph vertex generic types. Application tool types mirror the specialization of the artefacts types, with specializations of text viewer, list viewer, or graph viewer; each phase type grants access to application tool types;
- *an organizational view* (R7), with role and actor types;
- *a decision-oriented view* (R5), describing collaborative work at a fine grain level; each phase type is defined by a set of issue types, which must be solved either individually or collectively. Each issue type is characterized by a set of option types, which describe the different possible resolutions. At run time, one option is chosen through an argumentation process. Each option type can trigger, when it is chosen, some operation type, which can change some artefact, or change the process state (e.g. termination of a phase), or change the process definition itself.

### 3.2 Dynamic Task Model Refinement

A model designer can have to describe several strategies for performing the task. In our system, each strategy is described within a dedicated phase type. The choice between them takes place at run time, within a model refinement phase. The model refinement phase type includes a model refinement issue type, *with one option type for each strategy*. At run time, one of these option is chosen (either individually by someone playing a ‘process manager’ role, or collectively). The corresponding strategy is deployed, by instantiating new phase instances and new phase precedence (R4). By this way, artefact and process elaboration are similar, and the system can provide the same kind of assistance and guidance for both aspects.

### 3.3 Argumentation and Decision Support

Participants can express arguments about the different options of an issue and qualitative preferences between arguments. At every stage of the argumentation, the system computes a preferred option, *through an argumentative reasoning technique* similar to those described in [2, 3]. The issue, its options, the arguments ‘for’ and ‘against’ the options, the arguments ‘for’ and ‘against’ the arguments form an argumentation tree. A score and a status (active, inactive), which derive from the score, characterize each node of the tree. The score of a father node is the sum of the weights of its active child nodes which are ‘for’ the father minus the sum of the weights of its active child nodes which are ‘against’ the father. If the score is positive, the node becomes active, otherwise, it becomes inactive. Only status propagates in the tree because scores have no global meaning. Without preferences constraints all nodes have the same weight. Leaves are always active. The preferred option computed by the system has the maximum score. Preference constraints are qualitative preferences between two arguments: the importance of one argument is compared with the importance of the other. The aim of constraints is to define different weights for arguments in a very flexible and revisable way (R6). To each constraint is attached a “constraint issue” with three positions: ‘More important than’, ‘Less important than’, ‘Equally important than’. Thus, all constraints are subject to debate and can change their meaning dynamically. A constraint is active if both its source and its destination arguments are active, if one of its option is chosen (with a score strictly higher than the others), and if it is consistent with the other constraints. Constraint consistency is evaluated (with a path consistency algorithm) each time a constraint is created or becomes inactive. Weights are computed by a heuristics which gives a medium value for arguments not participating to constraints, a higher value for arguments which dominate other arguments and a lower value for arguments which are dominated by others.

Argumentation is useful both for individual and for collective issues. For individual issues arguments explain some rationale (e.g. in the demonstrative example, an individually proposed solution should come with a detailed rationale; on the opposite, a new constraint could come with a single argument as its justification). For collective issues argumentation prepares the resolution. In most cases the decision is kept separated from the argumentation result. Each issue type has a resolution mode property, which defines how issue instances are solved. For instance, ‘autocratic without justification’, when an actor playing a given role can take his own decision, ‘autocratic with justification’, when the decision maker has to argue in order to make his/her choice equal to the preferred choice of the system, ‘democratic’, when the preferred option computed by the system is chosen.

### 3.4 Synchronous and Asynchronous Work Support

In a typical deliberation, some result is negotiated collectively. At the beginning of the process, participants can express their opinion asynchronously. Synchronous sessions are useful when the argumentation becomes complex and when no rapid convergence occurs. Our deliberation system supports both working modes and provides adapted

guidance and awareness. The model designer can customize the way synchronous sessions are organized. First, synchronous sessions can be managed as artefacts, i.e. created through an issue resolution, by a given role, and accessed by the other participants through application tools (meeting list viewer, meeting reminder). Secondly, the organization process can be supported through issue types such as ‘Call for organization’, ‘Answer organization’, and ‘Call for participation’: the operation types that are triggered by these issue resolutions can use direct communication channels (R10) (e.g. dynamically built e-mail messages, internally managed by the system). This organization process can be described in a ‘Meeting organization’ phase type whose instances can run in parallel with other asynchronous phases. The system also provides *awareness indicators (R12) for deciding which issues require a synchronous discussion*. These indicators are computed through an analysis of the valuated argumentation tree and are directed towards participation (who has already participated? when?), divergence of opinions (who supports which option? by correlating participant interactions and option score evolutions), stability (is the preferred option computed by the system stable or not?). During asynchronous sessions, another awareness facility answers the ‘what’s new?’ question, highlighting entities that have changed since the end of the previous connexion of the same user. During synchronous sessions, several synchronous communication channels (chat, whiteboard, vote widget) are provided (R10). They can be used for instance for ‘meta discussion’ about the session (e.g. decision to skip to the next issue).

## **4 DOTS Prototype**

### **4.1 DOTS Architecture**

DOTS (‘Decision Oriented Task Support’) is our current prototype. It is completely developed in java for portability reasons (R8) and runs on the Internet (R9). It has a hybrid architecture with a traditional client/server architecture for asynchronous work and direct inter-client communication for synchronous work. The server is built on top of an open source multi user object base ([www.sourceforge.net/projects/stored\\_objects](http://www.sourceforge.net/projects/stored_objects)), which stores each model and all the corresponding instances (R11). The synchronous infrastructure (sessions, channels, clients, tokens) is provided by JSST ([www.java.sun.com](http://www.java.sun.com)). During synchronous sessions, one user (with the pre-defined ‘Moderator’ role) extracts already asynchronously started argumentations from the database and stores their results after the synchronous discussions.

### **4.2 DOTS Development Environment**

DOTS comes with a complete development environment. All tools are developed in java. First, a deliberation task model is written with DOTS modelling language and DOTS Integrated Development Environment (IDE). This model is compiled into java classes by the IDE. Together with the java classes of the generic kernel, they consti-

tute a dedicated environment. All these classes are compiled with a java compiler and a persistence post processor to become a persistent executable environment. Thanks to DOTS Instantiation tool, initial objects are created in the object base (actors, deliberation instances, initial documents, etc). The instantiated model can also be checked for completion by DOTS static analyser. Execution can start through the asynchronous client, which is independent of any deliberation model. The synchronous client is called from the asynchronous client. End users only manipulate the clients and the instantiation tool. Development of new deliberation models is performed by specialists (tool providers), who master the task modelling language.

### 4.3. DOTS Usage

#### The deliberation model – the process view

We explain DOTS usage through a collaborative document inspection model. Fig.1 shows the instantiated model. The ‘Initial document production’ is an individual phase instance, where an initial version of the document is produced. In the ‘Review model refinement phase’, the ‘process manager’ (or all the participants) choose dynamically how to perform the first review. In Fig.1, the first refinement defines a structured review beginning with parallel private reviews. The ‘Collective defect evaluation phase’ starts in asynchronous mode, and terminates with a synchronous session (virtual meeting) for discussing the more controversial defects. Its organization is defined in a dedicated phase running in parallel (AND operator). The ‘Document revision phase’ is individually performed by the author and produces a new version of the document. The second refinement defines a simpler process starting with the organization of a synchronous public discussion of the possible remaining defects. The third refinement terminates the deliberation.

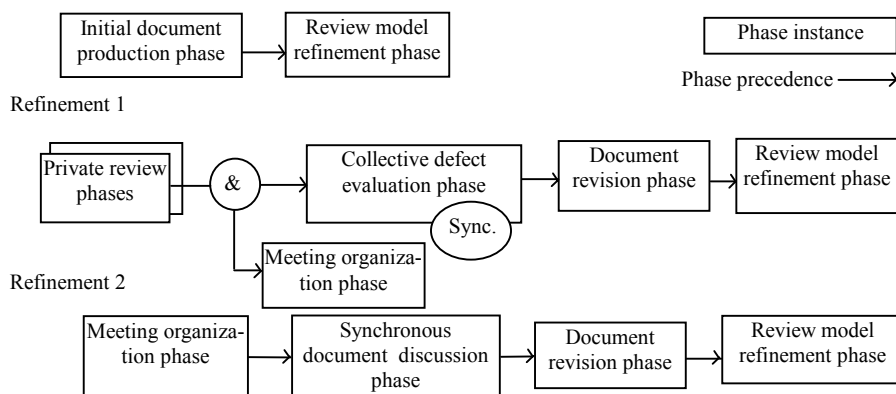


Fig.1. The process view

### The deliberation model – the decision view

Each phase type is basically defined by the issue types it contains. Table 1 summarizes the main issue types of the model. The most important one is the ‘Challenge defect’ issue type, where individually proposed defects are collectively discussed by all the inspectors. Individual issue types can have a single option type or multiple option types. In the latter case, the issue resolution maps an individual choice, the arguments being the rationale of the choice. In the former case, the resolution can either be completely automatic, as soon as the issue is raised (this case corresponds to an individual action), or can require at least one argument, as an action rationale. Collective issue types have at least two option types for discussion. The way to solve the issue is defined through the resolution mode defined in the task model. For instance, the ‘Challenge defect’ issue type can have an autocratic resolution mode (for the ‘process manager’ role), and the ‘Terminate Phase’ issue type of the ‘Collective defect evaluation’ phase can have a democratic mode.

Artifact types managed by this process include at least ‘Document’, ‘Defect’, and ‘Correction’. Several application tool types are available, such as ‘Document viewer’, ‘Proposed defect viewer’, ‘Accepted defect viewer’, ‘Rejected defect viewer’, ‘Correction viewer’, and perhaps a ‘Check list viewer’. All of them are specializations of ‘Textual viewer’ and ‘List viewer’ generic types. Finally, role types include ‘Process manager’ (the deliberation manager), ‘Inspector’, and ‘Author’.

Phase types	Issue types	Characteristics	Option types	Comments
Review model refinement	Choose refinement	Individual (process manager) or collective	Choose model 1, Choose model 2, ..., Terminate	Dynamic model refinement
All phase types	Terminate phase	Collective or individual	Continue, Terminate	Phase termination discussion
Private review and Collective defect evaluation	Propose defect	Individual	-	Defect proposal and description
Collective defect evaluation	Correlate defects	Collective (inspectors)	Correlate, Dissociate	Defect similarity discussion
Collective defect evaluation and synchronous discussion	Challenge defect	Collective (inspectors)	Keep, Reject	Defect discussion
	Suggest correction	Individual	-	Defect correction proposal
Meeting organization	Call for organization	Individual (process manager)	-	Propose some possible dates
	Answer for organization	Individual (inspectors)	-	Give preferences
	Call for participation	Individual (process manager)	-	Define the final date

Table 1. Main issue types

## 5 Conclusion

In new forms of distributed organisations, many issues are discussed through deliberation processes. These processes can vary in complexity and formality. This paper describes a comprehensive and generic support for a wide range of deliberations types

within distributed teams, which integrates concepts and techniques coming from workflow management systems, synchronous and asynchronous groupware, argumentation and decision making systems.

DOTS may supersede many previous systems, such as argumentation tools [2, 3], inspection tools [6], generic inspection environments, such as CSRS [9] and ASSIST [7]. CHIPS [1] is the closest prototype from DOTS that we know, mixing process support and collaboration support. CHIPS has a broader scope (flexible business processes), but provides no specific support for argumentation and decision.

Several aspects of DOTS need further work. First, the integration with distributed document management system on the Internet should be considered. Currently, DOTS provides no support for managing the shared artifacts. Secondly, if reuse of artifacts and deliberations is theoretically possible by keeping all terminated deliberations in the object base, we feel that a better alternative could be to feed the enterprise memory with these deliberations, in a more human readable form. A bi-directional interface with enterprise memory systems (for instance based on XML technologies) could be a very promising extension to the current system. Finally, we must assess the system with real users. A previous version has already be evaluated by students playing code inspections [5]. Now we are working with specialists of cognitive ergonomics for modeling evaluation of solutions during co-design processes [8]. We plan to produce a dedicated environment by parameterizing our generic kernel with such a model.

## References

1. Haake, J.M., Wang, W.: Flexible support for business processes: extending cooperative hypermedia with process support. *Information and Software Technology*, 41, 6 (1999).
2. Karacapilidis, D., Papadias, D.: A group decision and negotiation support system for argumentation based reasoning. In: *Learning and reasoning with complex representations*, LNAI 1266, Springer-Verlag, Berlin Heidelberg New York (1997).
3. Karacapilidis, D., Papadias, D., Gordon, T.: An argumentation based framework for defeasible and qualitative reasoning. In: *Advances in artificial intelligence*, LNCS 1159, Springer-Verlag, Berlin Heidelberg New York (1996).
4. Lonchamp, J.: A generic computer support for concurrent design. In: *Advances in concurrent engineering*, CE2000, Lyon, Technomic Pub. Co, Lancaster, USA (2000).
5. Lonchamp, J., Denis, B. : Fine grained process modelling for collaborative work support, *Journal of Decision Support Systems*, 7, Hermes, Paris (1998).
6. Macdonald, F., Miller, J., Brooks, A., Roper, M., Wood, M.: A review of tool support for software inspection. *Proc. 7<sup>th</sup> Int. Workshop on CASE* (1995).
7. Macdonald, F., Miller, J.: Automatic generic support for software inspection. *Proc. 10<sup>th</sup> Int. Quality Week*, San Francisco (1997).
8. Martin, G., Détienne, F., Lavigne, E.: Negotiation in collaborative assessment of design solutions: an empirical study on a Concurrent Engineering process. In: *Advances in Concurrent Engineering*, CE2000, Lyon, Technomic Pub. Co, Lancaster, USA (2000).
9. Tjahjono, D. : Building software review systems using CSRS. *Tech. Report ICS-TR-95-06*, University of Hawaii, Honolulu (1995).