Enhancing Dependability of Cooperative Applications in Partitionable Environments *

François J.N. Cosquer — fjnc@inesc.pt — IST[†]INESC[‡]

Pedro Antunes — paa@inesc.pt — IST-INESC

Paulo Veríssimo — pjv@di.fc.ul.pt — FC/UL[§]

Abstract

This paper presents a pragmatic approach to providing partition processing system support for cooperative applications. A method for specifying and programming application-level partition processing strategies is described. The system support is based on a partition typing mechanism which allows the application programmer to model the relative importance of partitions. This is combined with a split/merge rules configuration table through which the partition processing strategy is defined. In the context of cooperative application semantics, our approach combines the correctness of the pessimistic, and the availability of the optimistic approaches for data management in partitionable environments. The paper focuses on the practical issues linked with, firstly, the specification, and secondly, the support at runtime, of the partition processing strategies. This approach is relevant in the context of large-scale asynchronous distributed systems such as the Internet, which, as a result of current technology and topology, are inevitably prone to partitions. Examples are given, illustrating how the partition support is used and combined with new feedback techniques in order to implement more robust cooperative environments.

Keywords: *Fault-Tolerance, Cooperative Applications, Tool, Configuration, Interconnected Networks.*

*This work was partially supported by the CEC, through Esprit Project BR 6360 Broadcast and CaberNet, Basic Research Network of Excellence 6361 in Distributed Computing Systems. **This paper appeared in the proceedings of the 2nd European Dependable Computing Conference, Italy, October 1996**

[†]Instituto Superior Técnico - Technical University of Lisboa

[‡]Instituto de Engenharia de Sistemas e Computadores, R. Alves Redol, 9 - 6º - 1000 Lisboa - Portugal, Tel.+351-1-310000.

[§]Faculdade de Ciências da Universidade de Lisboa. The Navigators Group Web page: http://www.navigators.di.fc.ul.pt/

1 Motivation

The growth of interconnected networks has enabled the development of a wide range of distributed applications. Among the most promising are those dealing with multi-user collaboration, usually referred to as Computer Supported Cooperative Work (CSCW) or groupware. Cooperative applications include tools as varied as shared-drawing, concurrent engineering, teleconferencing, etc. These cooperative tools have a multitude of application fields including education, business, medicine, etc.

Cooperative applications represent a high demand for large-scale distributed systems because they lessen the need for a collection of people to commute to a single location. Furthermore, the widespread existence of simple cooperative tools has broken the physical distance barrier and enabled daily interactions over large distances. However, large-scale communication infrastructures suffer from problems inherent to the nature of the current technology, topology, and, of course, sheer dimension and distance. One obvious problem is that they are prone to partitioning, meaning that sites may occasionally be unable to communicate with each other. Furthermore, inaccurate failure detection due to the high variance and unpredictability of communication delays may also lead to what are known as "virtual partitions"[1].

These problems not only represent an obstacle to the development of new cooperative applications, but also, to a wider use of existing applications. The symptoms they generate burn down to one factor: dependability impairments. In fact, network partitioning hinders the correct execution of any application that shares state, be it messages or data. The major sources of unreliability are: blocking upon partitions, and/or inconsistency upon uncontrolled mergers.

Surprisingly, having surveyed a number of groupware applications and platforms[2], we have found little or no concern for scaling and fault-tolerance issues in general. Most applications are based on the "absence of partitions" assumption[1] which, we believe, is not justified when targeting current asynchronous distributed systems. There exist a number of partition processing techniques which have been developed for database and filesystem applications[3, 4]. Such techniques address data consistency and availability issues. Two extreme approaches for these techniques are the *pessimistic* approach and the *optimistic* approach. The former trades availability for correctness while the latter allows availability of data but might have to resolve conflicts upon reconnection.

These techniques in their current form do not solve all problems of collaboration-oriented applications. Cooperative applications require a notion of what is usually referred to as *collaboration awareness* at all levels, hence the need for a notion of partition at the programming level. *Partition support*, as used in this text, means preserving the notions of control, coordination and cooperation.

In [5], we have pointed out the role of the functionality provided by group-oriented systems and the requirements of groupware applications. Early group-oriented systems, like Isis[6], only allow a majority partition to make progress, minority partitions having to abort. Recent membership protocols are promising because they integrate the notion of multiple partitions [7, 8].

Another motivation for our work came from connectivity statistics of the Internet, the largest distributed computing infrastructure today. It was observed on some links that 95% of partitions do not last on average more than a few minutes[1], a duration which is too high to be ignored and too low to reschedule the cooperative session taking place.

The above considerations have led us to believe that dependability of cooperative applications would highly benefit from the underlying system support if the latter could:

- Allow multiple partitions to operate simultaneously.
- Control progress in each partition, according to a correctness specification.
- Provide connectivity feedback clues to users.

The first two objectives are normally antagonistic: they correspond respectively to the aforementioned optimistic and pessimistic policies. The paper describes an approach for attaining both objectives and thus, contributes to bridging the gap between application requirements and current limitations of large-scale asynchronous computing systems.

In a brief explanation, we abandon the primary-partition paradigm, whose liveness— and thus availability— can be compromised in a large-scale setting, since several partitions form many often, not rarely none of them being primary. In that sense, our approach could be termed optimistic. However, we exploit the conjunction of two attributes, namely: the low rate of conflict occurrence due to collaboration awareness superimposed on raw data operations; and the assumption of short-lived connectivity disruptions. In consequence, we are capable of defining a semantics that yields the global correctness typical of pessimistic approaches, whereas we achieve a virtually non-blocking operation, as seen in optimistic approaches. In that sense, we termed our approach *pragmatic* partition processing. At runtime, the user is aware of the occurrence of partitioning. However, the system support allows seamless transitions between the various connectivity situations. By allowing multiple partitions to operate simultaneously, the system support enables applications to proceed during transient disruptions in connectivity according to the specified semantics. The third objective, of connectivity feedback, is achieved through the instrumentation (e.g. failure detection) we put in place to help materialize our pragmatic semantics. The interested reader will find a more detailed description in [9]. We

[1] From monthly statistics reports of the T3 backbone in 1994.

believe this work to be an innovative experiment in distributed applications in general, and in groupware system support in particular.

The paper is organized as follows. Section 2 briefly presents the membership problem and how partitions are dealt with in group-oriented systems. Section 3 presents NavCoop, a groupware platform which constitutes the framework for the work presented in this paper. It describes PSS, the Partition Support Service, the NavCoop module which implements the partition processing support. Section 4 illustrates the applications and our current experience with PSS. Section 5 presents the concluding remarks.

2 Group-Oriented Systems and Partitions

Group-oriented systems can be seen as providing two main services:

- a *membership service* which maintains consistent information, called the *view*, about which processes are involved in a distributed computation at any given time. The membership protocol forces the system to conform with connectivity information provided by what is referred to as the *Failure Detector* (FD).

- a *multicast service* which provides various semantics of group communication, such as for example, causal and total ordered multicast[10, 11].

A particular combination of the two services implements what is usually known as Virtual Synchrony (VS)[6]. However, group-oriented systems suffer from the fact that failure detectors (FD) are unreliable due to asynchronism in the underlying system (a result whose foundation lies in the FLP impossibility result[12]). In practice, this has been partially overcome by early systems such as Isis, whose FD is tuned to make few mistakes in Local Area Network (LAN) environments. However, such systems can still partition in case of erroneous suspicions or broken communication links: they implement what is known as the primary partition strategy. The system identifies at most one partition where some pre-defined condition holds (usually, the majority of processes) and prevents progress in any other partition by forcing processes to leave the system.

The primary partition offers a simple, yet efficient, solution to a wide range of distributed applications[13]. This is especially true in LAN environments because partitions are rare events. However, the primary partition approach implements a behavior which does not satisfy the initial requirements presented in Section 1. For example, the system might block as long as the majority condition is not satisfied. Furthermore, the system forces members of minority partitions to quit the application.

Recently, new membership protocols have been developed which take into consideration the problems associated with large-scale distributed systems. This means that the membership

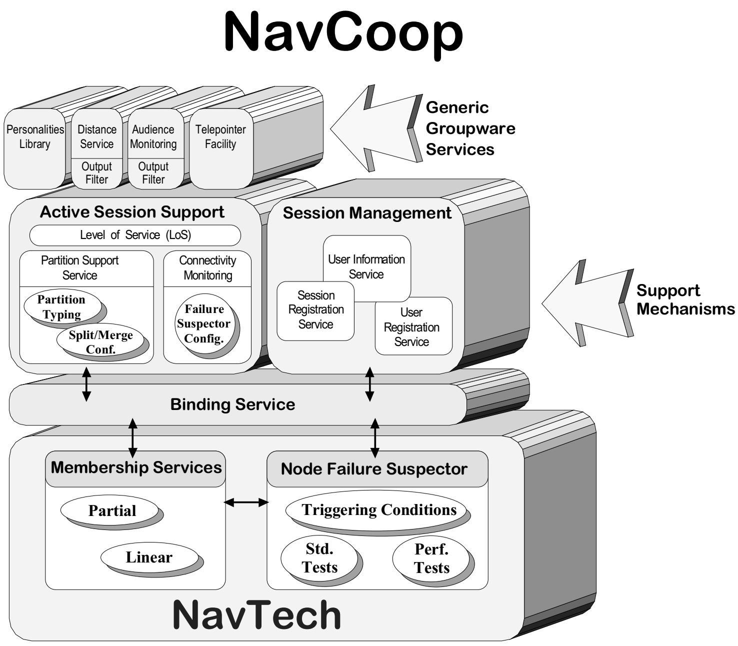

Figure 1: NavCoop functional architecture

service will deliver (concurrent) views in different partitions. In[7], a protocol which allows multiple partitions to operate simultaneously offers a service referred to as *strong-partial membership*. Informally, this means that the service guarantees that, at any time in the system, concurrent views are non-intersecting. Partial membership services are being implemented in group-oriented systems like Horus[14], Totem[15] and Transis[16]. Finally, [17] proposes a fully configurable membership protocol. The customized membership service is built by composing separate modules, each implementing some abstract property.

Some systems try to preserve the consistency properties of virtual synchrony at low level, across partitions. This is the case with the work on extended virtual synchrony[18]. Our research effort on groupware support has been directed towards relying on a *partial membership service* at the communication level, ensuring virtual synchrony as the baseline communication paradigm. Consistency in the presence of partitions is achieved by supplying a semantics of partition to the programmer.

3 The NavCoop Partition Support Service

3.1 NavCoop Overview

NavCoop is a groupware platform developed at INESC which is intended to provide support mechanisms and a set of generic groupware services dedicated to large-scale networks.

NavCoop is based on a group-oriented communication architecture as shown in Figure 1. An earlier phase of our work emphasized the need for configurable failure suspectors[9]. More recently, efforts have been devoted to programming and runtime support mechanisms for partition processing. PSS, Partition Support Service, is the NavCoop module which provides support for partition processing. PSS is based on two concepts: partition levels and the associated split/merge rules configuration interface.

3.2 Terminology and System Considerations

In order to avoid possible confusions, we clarify below the key concepts involved with the Partition Support Service.

- *Group and sub-group:* a group is a set of participants involved in a computation. A sub-group is a sub-set of the participants in a group originated by the splitting of a group (typically due to a partition). The semantics chosen for the membership service implies that concurrent sub-groups are disjoint.
- *Partition:* designates a system event that leads to the splitting of a group in two or more sub-groups.

PSS is based on the following assumptions about the underlying system.

- *Fault-coverage:* the targeted fault-coverage of PSS are link failures. Links may fail by not delivering data (omission failure). The intention is to cover temporary disruption in connectivity between nodes (i.e. recoverable). Process crashes may be masked using traditional redundancy techniques[19, 20].
- *Dynamic join and leave operations:* the underlying group-communication subsystem allows voluntary "join and leave" operations. The system also differentiates those operations from failure suspicions which lead to the splitting of the group of processes involved in a computation into sub-groups.

3.3 Levels and Typing

Users interested in participating in a cooperative application first have to register their intention. The list of registered users will be the basis for partition processing support. It is referred to as the *initial* group as it corresponds to a partition-free set-up.

The NavCoop PSS implements *levels*. The levels, $[1, 2, ..., n]$, reflect the relative importance of the group/sub-groups. Level 1 corresponds to the *initial* group, level 2 to the second most important, and so forth. The depth, n, is dictated by the needs of the application, as will become clearer ahead. Each level is specified by means of type and composition definitions:

Level($l, \langle$ *type, composition* $\rangle$) - specification of the *type* and *composition* of the group/sub-group of level l.

Typing applies the idea of social role or *role*, a concept found in cooperative applications to model the relative importance of each user. The role is materialized by two variables: the *identity* of a participant, and its *weight*[2]. In PSS, weight values reflect the importance of users, both during the normal operation of a cooperative session, and during partitions. Identities are used to designate group/sub-groups containing given participants. The *composition* of a group/sub-group $\mathcal{G}$ is defined by its m participants p_i, and the sum of the individual weights $w(p_i)$. The total sum of weights in the *initial* group is referred to as $\mathcal{W}_{ini}$. The three resulting *type* and *composition* specifications are listed below.

- **Quorum group/sub-group:** specifies that a group/sub-group comprises a known sum of weights. A group/sub-group $\mathcal{G}$ is defined as a quorum Q group/sub-group iff $[\sum_{i=1}^{m} w(p_i) > Q]$. Q is a pre-defined value.

- **Designated group/sub-group:** specifies that a group/sub-group comprises a set of known user(s). A group/sub-group $\mathcal{G}$ is defined as a designated D group/sub-group iff $[D \subseteq \mathcal{G}]$. D is a pre-defined set of participants $\{p_j, ..., p_u\}$.

- **Combined group/sub-group:** specifies that a group/sub-group comprises a known sum of weights Q and a set of designated user(s) D. A group/sub-group $\mathcal{G}$ is defined as a combined $Q - D$ group/sub-group iff $[\sum_{i=1}^{m} w(p_i) > Q \wedge D \subseteq \mathcal{G}]$. Q is a pre-defined value. D is a pre-defined set of participants $\{p_j, ..., p_u\}$.

Formally speaking, the importance of group/sub-groups, expressed by the levels, is translated into progress specifications:

Progress($l, \langle$ *progress* $\rangle$) - specification of the *progress* of participants in a level l group/sub-group.

The main idea behind decreasing importance is to capture a degradation of the level of activity a participant is allowed to have, as it gets further away from the initial group of participants. In classical approaches, progress specifications are essentially binary: simply allowing all reads and writes (as in a primary partition), or blocking all activity (as in a minority partition). With the mechanisms we introduced however, we can define richer semantics, such as proceeding *up-to* a certain application phase, or execute *only* a subset of actions.

[2] Weights have also been proposed in [21] for maintaining replicated data.

As soon as the application starts running, the Level(l) is evaluated in increasing order of l (1→n), to determine which level does a given group/sub-group belong to. Registered users start in the level corresponding to their current connectivity, obeying the relevant semantics Progress(l).

It is up to the application programmer to define the level depth n, and to make the several Level(l) and Progress(l) specifications. These mechanisms are complemented by a configuration interface called the split/merge rules table, which allows the programmer to specify how the application should respond to a partition event at runtime.

3.4 Split/Merge rules configuration table

When the application is running, the group communication subsystem forwards *views* to the NavCoop platform from which the new level is evaluated. The NavCoop level event can be described as one of the following:

$E_{downgrade(k)}$: The current level was decremented by k levels.
$E_{upgrade(k)}$: The current level was incremented by k levels.

To each of those events corresponds an application level rule referred to as *split/merge rule*. The application programmer defines the code corresponding to each rule as a way to implement the desired behavior. The split/merge functions are presented in Table 1. The number of rules to be defined for a level depth of n is $n * (n-1)$. This number might appear high at first glance, but in practice a number of the functions will be identical and/or share code. In any case, there will always be a tradeoff involved between the programming complexity of the support, and the functionality enhancement provided to the users at runtime. However, the services provided by a groupware platform such as NavCoop may drastically simplify the application programmer's task in what concerns dependability in face of partitions.

3.5 Current Status

NavCoop is based on the NavTech communication subsystem. Following the xAMp protocol suite[22], NavTech is a new group-oriented communication subsystem developed at INESC which is intended to support the development of reliable applications over large-scale networks[23].

The current prototype of NavCoop was developed using Horus-based virtual synchrony[14], NavTech configurable failure suspector and abstract network layer. NavCoop is implemented in C, uses the OSF Motif toolkit, and runs over a network of UNIX workstations. NavCoop mechanisms and the Horus/NavTech protocols run in the same Unix process.

Levels	Transition event	
	$E_{downgrade(k)}$	$E_{upgrade(k)}$
1	`func`($split_{1\to 2}$) `func`($split_{1\to 3}$) . `func`($split_{1\to n}$)	not applicable
2	`func`($split_{2\to 3}$) . `func`($split_{2\to n}$)	`func`($merge_{2\to 1}$)
.	.	.
n	not applicable	`func`($merge_{n\to 1}$) `func`($merge_{n\to 2}$) . `func`($merge_{n\to n}$)

Table 1: Split and Merge rules table

4 Applications and Experience

Terminology. This work bearing relationship with two research fields that use the same words with different meanings, we are left in the uncomfortable position of having to make a differentiation. Since we have already used "synchronous" and "asynchronous" with the traditional meaning taken in the distributed systems community— whether time bounds exist and are known, or otherwise— we will adopt other terms for synchronous and asynchronous cooperation as used by the CSCW community. Without any attempt to create a new terminology, we will use in this paper "synchronized" and "non-synchronized", when referring to operations performed simultaneously (i.e at the same time) or independently (i.e at different time) by users.

We present below two examples which illustrate the benefits of the NavCoop PSS. The first example illustrates how well-accepted groupware artifacts used for synchronized cooperation can be modified for providing a partition-resilient behavior. The second example briefly relates our experience with a "same time/different place" Group Decision Support System showing how partition support can be integrated in already existing applications.

4.1 Telepointer facility

When designing the NavCoop generic services (see Figure 1), we studied and experimented the way of incorporating feedback in various building blocks. In order to further improve the level of support at runtime, we explored how connectivity and partition level information could be integrated in existing artifacts. The idea consisted of combining the typing information provided by the PSS with standard cooperative modules using new feedback techniques for improving group awareness. To illustrate our claim we have selected here a mechanism frequently used during synchronized cooperative sessions: the *telepointer*, which is used for gesture support.

A telepointer is a shared graphical entity which is manipulated by one participant at a time in order to point or get attention to a specific area or item of the shared workspace. Telepointer facilities are usually provided by many synchronized cooperative applications. The telepointer usually assumes that position, movements and timing of the telepointer are shared by all users. The aim is to make the telepointing mechanism rich enough to express a user's intentions to the whole group.

The basic design problem we address concerns the implementation of a WYSIWIS paradigm (What You See Is What I See) in presence of long communication delays which may eventually lead to partition. Traditional straightforward implementations suffer from the underlying system limitations. The contradiction between users' expectations and system limitations results in lost of shared context and ambiguity.

The principles adopted for the telepointer are illustrated in Figure 2 consist of the following three steps.

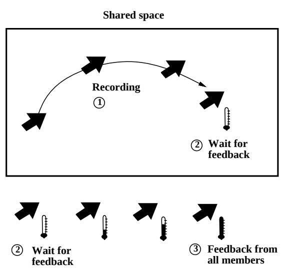

Figure 2: The Telepointer

1. The first step avoids flooding the network with too many messages. The telepointer movement is recorded until the user stops. The recorded movement is then distributed to other participants in one single message.

2. In the second step, when each participant's shared space receives a recording, it executes a play-back, i.e., replays the movement. Furthermore it broadcasts an acknowledgement message to the group informing that the operation is completed.

3. In the third step, we use the count of feedback messages from the group participants to inform the telepointer's user of the current shared context. The feedback information is displayed using a meter located next to the telepointer (see right part of Figure 2).

After recording a gesture, the user should refrain from moving the telepointer until knowing that all participants have replayed the gesture. This information is provided by the meter icon. After the meter goes full, the user is sure that all participants have seen the telepointer movement.

This functionality was implemented using the communication service with acknowledged message deliveries.

The design problem related to partitions is to define which degree of telepointer semantics it is possible to sustain during a partition. Our strategy is to proceed with telepointer usage *but* to inform users that not all members will be aware of the recorded gestures. The functionality of the telepointer is maintained in partition mode but the participants are notified that not all original group members are participating. We have defined a generic scenario in which there exists a unique *major* sub-group and possibly many *minor* sub-groups.

1. A *major* sub-group holds the telepointer owner and therefore sustains the telepointer semantics to reachable members;

2. A *minor* sub-group which does not hold the telepointer owner. A *temporary* telepointer owner is designated in order to preserve *localized* telepointer semantics.

The corresponding levels were defined using PSS and are given in Table 2. Recall that levels are evaluated in increasing order $[1 \rightarrow \mathrm{n}]$. The default policy based on this information allows each sub-group to possess its own telepointer so as to preserve animation during partition mode. The users are immediately aware of the situation by seeing a shadowed telepointer. This is illustrated in the top half of Figure 3. This Figure also shows that the telepointer is displayed differently to owners/not owners. Also note that, if excluded participants have not crashed, they also have created their sub-group and entered into partition mode. It was decided to implement three different graphical aspects for telepointer owner depending on the level (black for *initial*, black shaded grey for *major* and grey shaded black for *minor*). For not owners, the telepointers is displayed in white for the *initial* group and shaded grey in both the *major* and *minor* sub-groups.

Level	**Type**	**Composition**	**Progress**
1-initial	Quorum	$Q = \mathcal{W}_{ini}$ (all reg. users)	unrestricted
2	Designated	$D =$\{telepointer owner\}	unrestricted
3	Designated	TRUE	temporary owner

Table 2: Telepointer PSS Levels

When participants who were excluded from the *major* sub-group become reachable again, the system will terminate the partition mode associated with the *minor* sub-group. The system merges and eliminates any condition local to *minor* sub-groups. We defined in Table 3 the different merge functions that may occur. We detail below the various merge cases (see bottom half of Figure 3.

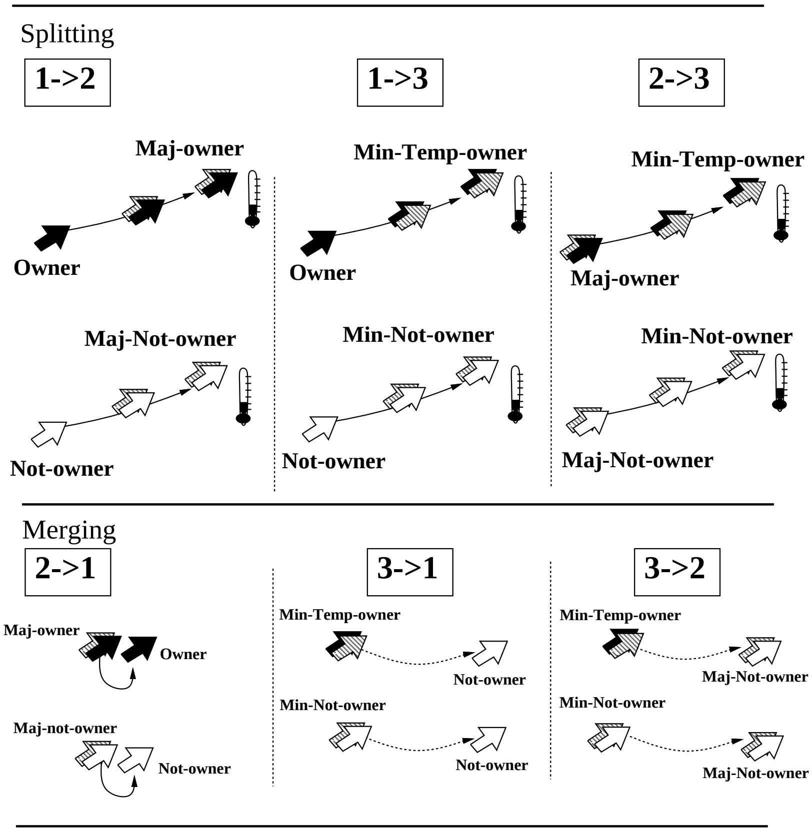

Figure 3: Partition mode feedback

- **2 → 1:** corresponds to the *major* sub-group merging back to the *initial* group.
  - The owner: the telepointer owner is a user in the major sub-group and the system decides that it will continue to own it. The telepointer regains its original graphical aspect and remain in same location.
  - Not owner: for all other users who are not owner of the telepointer, the telepointer just regain its graphical aspect.
- **3 → 1:** corresponds to a *minor* sub-group merging back to the *initial* group.
  - The owner: the temporary telepointer owner of the *minor* sub-group looses its privilege and the graphical aspect is changed back to a not owner telepointer. If the temporary telepointer had been used it moves (animation) to the new owner's telepointer location.
  - Not owner: the telepointer just regain its original graphical aspect with a possible animation back to the new owner's telepointer location.
- **3 → 2:** corresponds to a *minor* sub-group merging to the *major* sub-group.
  - The telepointer owner: the telepointer was owned by a user in a *minor* sub-group. The system will give the telepointer to the owner in the *major* sub-group. The telepointer changes its aspect and moves (animation) to the other participant's telepointer location.
  - Not owner: the telepointer keep the same graphical aspect with a possible animation to the current *major* sub-group owner position.

- **3 → 3:** corresponds to 2 *minor* sub-groups merging.
  - The telepointer owner: The system will give the telepointer to the oldest owner of the 2 *minor* sub-groups.
  - Not owner: the telepointer keeps the same graphical aspect with a possible animation to the current selected sub-group owner position.

Levels	**Transition event** $E_{downgrade(k)}$	$E_{upgrade(k)}$
1	`func`$(split_{1\to2})$ { Shadow telepointer() } `func`$(split_{1\to3})$ { Designate temporary owner() Owner record, others playback() Shadow telepointer() }	
2	`func`$(split_{2\to3})$ { Designate temporary owner() Owner record, others playback() Shadow telepointer() }	`func`$(merge_{2\to1})$ { Owner record, others playback() Remove shadow() }
3		`func`$(merge_{3\to1})$ { Disable temporary owner() Move to owner's position() Remove shadow() } `func`$(merge_{3\to2})$ { Disable temporary owner() Move to owner's position() Remove shadow() } `func`$(merge_{3\to3})$ { Change temporary owner() Move to owner's position() }

Table 3: Split and Merge rules table

In our telepointer implementation, users are aware of how a synchronized operation is being carried out. This functionality is complemented by the audience monitoring service which informs the participant of the current composition of his/her group/sub-group.

4.2 The NGTool Experiment

4.2.1 Application Overview

NGTool is a Group Decision Support System (GDSS) which runs on Unix workstations over the Internet and uses X Windows and the OSF Motif toolkit. The NGTool supports same time/different place meetings. The tool provides synchronized operations over a display space shared by 5/6 users, one of which known as the *moderator*. The functionality of the NGTool is described in [24, 25]. The NGTool was ported to the NavCoop PSS as depicted in Figure 4. We will illustrate the implemented functionality for a particular meeting situation.

Figure 5 shows screen dumps of the NGTool of two users, a participant on the left and the moderator on the right, in the phase when they are generating and proposing ideas. The ideas are

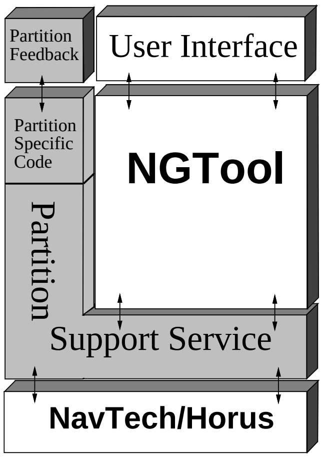

Figure 4: Porting NGTool to NavCoop

generated (i.e. written down on a text field) in a private space which is on the screen's left hand side. Ideas are represented by a small electric lamp. To propose an idea to the participants of the meeting, the user must drag the idea from the private space to the shared space: the screen's right hand side. To mediate this operation the NGTool provides a graphical entity named teleassistant ("switched-on" electric lamp). The teleassistant controls concurrent accesses to the shared space and coordinates the actions of the moderator and other users. Since the teleassistant provides feedback to all users over these events, it is a natural origin for delivering information about partitions.

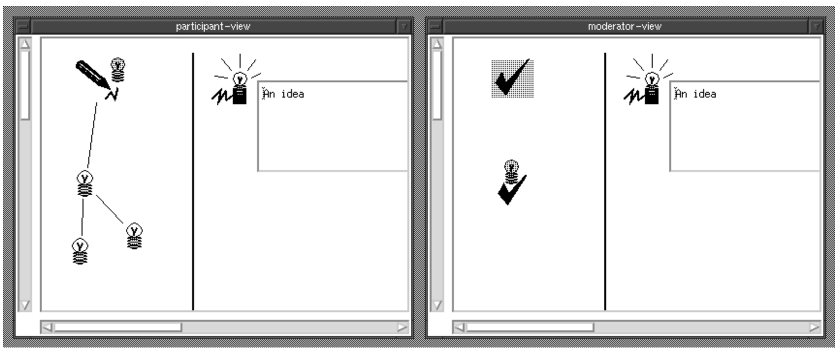

Figure 5: The NGTool windows

4.2.2 Adding Partition Functionality

When defining the partition processing strategy, a preliminary study led us to consider the following aspects:

- *Relative importance of sub-groups:* due to the moderated aspect of the NGTool session it was decided to discriminate two types of sub-groups. Those are referred to as *moderated* and *non-moderated* depending on whether the sub-group includes the moderator.

- *Temporal factor:* we identified two cases depending on the time-frame of the connectivity disruption. This is because the NGTool session goes through different phases. As we will discuss below this has consequences on the merging phase and rules definition.

The NGTool was modified such that when a partition occurs both individual and group work can proceed. We describe here the strategy adopted during the phase of generating and proposing ideas. As explained in Section 3, the first task is to define the levels (see Table 4). This definition means that the application will only differentiate two kinds of progress depending on the presence of the moderator. The NGTool designer made this decision since the application is strongly coordinated by the moderator. This results in the fact that moderated sub-group will always have higher privileges than non-moderated sub-groups.

Level	**Type**	**Composition**	**Progress**
1-initial	Quorum	$Q = \mathcal{W}_{ini}$ (all reg. users)	unrestricted
2 *(moderated)*	Designated	D ={moderator}	unrestricted
3 *(non-moderated)*	Designated	TRUE	same phase

Table 4: NGTool PSS Partitions Levels

When a partition occurs, the sub-group which includes the moderator is allowed to proceed work as if no partition had occurred, i.e. users can generate and propose ideas. However, this sub-group is visually informed of the partition occurrence, in order to be aware that not all of the original members are participating. Figure 6 illustrates the situation for the *moderated* sub-group, showing a small two-hands icon under the teleassistant. A sub-group which looses moderation is allowed to generate ideas (in the users' private spaces) The graphical partition feedback representation in the *non-moderated* sub-group is the opposite of the *moderated* i.e. grey hand with white shadow. However in order to comply with the original (non-computerized) NGT semantics, users in *non-moderated* sub-groups cannot propose ideas. We are considering future experiments with more optimistic semantics, such as letting non-moderated sub-groups propose ideas.

Figure 6: Partition feedback

When the system merges, it is assumed that the moderated sub-group has several ideas in its shared space which are not available to the previously disconnected, non-moderated sub-groups.

The merge function must therefore re-synchronize the shared space by delivering the missing ideas to the non-moderated sub-groups.

However, the temporal factor aspect influences the merge functions since the NGTool phase may have changed during the occurrence of the partition. Being isolated from the moderator of the session, members of the non-moderated sub-group(s) cannot change session phase. This means that it is possible that a merge occurs between a *moderated* sub-group and a *non-moderated* sub-group which are not in the same NGTool phase. As a result, the NGTool defined two types of merges depending on the phases of the session of the various sub-groups merging. PSS does not provide support for this kind of conditional merge function (referred to as temporal factor) but only level transition function definition. Therefore, the functionality was implemented as part of the application. This resulted in the configuration presented in Table 5.

Levels	Transition event	
	$E_{downgrade(k)}$	$E_{upgrade(k)}$
1	`func`$(split_{1\to2})$ { Show moderated partition icon() } `func`$(split_{1\to3})$ { Disable public teleassistant() Show non-moderated partition icon() }	
2	`func`$(split_{2\to3})$ { Disable public teleassistant() Show non-moderated partition icon() }	`func`$(merge_{2\to1})$ { Remove partition icon() }
3		`func`$(merge_{3\to1})$ { Enable public teleassistant() Check phase changes () Re-synchronize shared space() Remove partition icon() } `func`$(merge_{3\to2})$ { Enable public teleassistant() Check phase changes () Re-synchronize shared space() Show moderated partition icon() }

Table 5: Split and Merge rules configuration table

The main benefits of using the NavCoop platform for the development of the NGTool come from the approach to *pragmatic* partition processing support. Although levels and split/merge rules must be defined accordingly to the particular semantics of the application, NavCoop PSS allows tackling the problem associated with partitions in a structured and manageable way.

5 Concluding Remarks

The paper presented a method for defining, what we refer to as, *pragmatic* partition processing strategies, exhibiting the correctness of pessimistic, and the availability of optimistic approaches to the management of partitionable applications. Based on a typing mechanism which is combined with a split/merge configuration table, the Partition Support Service has been integrated in the NavCoop groupware platform.

The pragmatic partition support approach provides a concrete answer to application designers for tackling situations of temporary deviations from normal operation due to connectivity problems. It is known that these phenomena contribute to the lack of dependability of distributed applications, and of cooperative applications in particular. Observed application blocking and data inconsistency are current symptoms, and they should be addressed by adequate measures to tolerate partition faults. We proposed reconfiguration mechanisms to overcome the transient faults caused by partitions, based on the semantics of cooperative applications, namely, the social roles played by the participants. We believe these results to be innovative, with regard to approaches taken in other fields of work, such as distributed databases. The latter depended on lower level data operation semantics, and were thus less flexible, leading to the pessimistic versus optimistic dichotomy.

Our method forces/helps the application designers to carefully define the importance of participants taking part in a cooperative activity. The configuration process is intended to provide a solution for closely modeling the real-life cooperative activity. The benefits of this approach were illustrated by examples. At the implementation level, this work constitutes an experiment in using group communication systems services for supporting cooperative applications. Our experience shows how partial membership services can be used to enhance cooperative applications support.

Further work is underway to validate our initial results. We need to ensure that the complexity inherent to the various configurations proposed can be partially hidden from the application programmer. For example, we aim at simplifying the rules generation by making this process partially automatic. Overlooking this issue could be detrimental to the use of groupware support technologies.

Acknowledgements: The authors would like to thank Jorge Frazão and Nuno Guimarães, who, besides the authors, contributed to the two demos where the NGTool and NavCoop support were demonstrated; Broadcast Open Workshop (Grenoble, July, 1995) and the Cyted-Ritos International Workshop on Groupware (Lisboa, September, 1995). We are also grateful to Carlos Almeida, André Zúquete, David Matos, Susan Tinniswood, Alexandre Lefebvre and Luís Rodrigues for commenting on early versions of this document.

References

[1] Aleta Ricciardi, Andre Schiper, and Kenneth Birman. Understanding Partitions and the No Partition Assumption. In *Proceedings of the 4th Workshop on Future Trends of Distributed Computing Systems*, pages 354–360, September 1993.

[2] François J.N. Cosquer and Paulo Veríssimo. Survey of Selected Groupware Applications and Supporting Platforms. Technical Report 2nd Year - Vol. 1, BROADCAST, Rua Alves Redol 9-6°, 1000 Lisboa, Portugal, September 1994. (Also available as INESC Report RT-21-94).

[3] Hector Garcia-Molina Susan B. Davidson and Dale Skeen. Consistency in partitioned networks. *Computing Surveys*, 17(3):341–370, September 1985.

[4] James J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[5] François J.N. Cosquer and Paulo Veríssimo. The Impact of Group Communication Paradigms in Groupware Support. In *Proceedings of the 5th Workshop on Future Trends of Distributed Computing Systems, Cheju Island,*, August 1995.

[6] Kenneth P. Birman and Thomas A. Joseph. Exploiting Virtual Synchrony in Distributed Systems. In *11th Symposium on Operating Systems Principles*, pages 123–138, November 1987.

[7] A. Shiper and A. Ricciardi. Virtually Synchronous Communication based on a weak failure suspector. In *Proceedings of the 23rd Int. Conf. on Fault Tolerant Computing Systems*, June 1993.

[8] Danny Dolev, Dalia Malki, and Ray Strong. An Asynchronous Membership Protocol that Tolerates Partition. Technical report, Institute of Computer Science, The Hebrew University of Jerusalem, Israel, 1995.

[9] François J.N. Cosquer, Luís Rodrigues, and Paulo Veríssimo. Using Tailored Failure Suspectors to Support Distributed Cooperative Applications. In *Proceedings of the 7th International Conference on Parallel and Distributed Computing and Systems*, Washington D.C., October 1995.

[10] Luís Rodrigues and Paulo Veríssimo. Causal Separators for Large-Scale Multicast Communication. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, June 1995.

[11] L. Rodrigues, H. Fonseca, and P. Veríssimo. Totally ordered multicast in large-scale systems. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, pages 503–510, Hong Kong, May 1996. IEEE.

[12] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the Association for Computing Machinery*, 32(2):374–382, April 1985.

[13] Kenneth P. Birman and Robbert van Renesse. *Reliable Distributed Computing with the Isis Toolkit.* IEEE Computer Society Press, 1994.

[14] R. van Renesse, Ken Birman, Robert Cooper, Brad Glade, and Patrick Stephenson. The Horus System. Technical report, Cornell University, July 1993.

[15] Y Amir, L. Moser, P. Melliar-Smith, D. Agarwal, and P. Ciarfella. Fast Message Ordering and Membership Using a Logical Token-Passing Ring. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 551–560, Pittsburgh, Pennsylvania, USA, May 1993.

[16] Danny Dolev, Dalia Malki, and Ray Strong. A Framework for Partitionable Membership Service. Technical Report CS95-4, The Hebrew University of Jerusalem, Jerusalem, Israel, 1995.

[17] Matti A. Hiltunen and Richard D. Schlichting. A Configurable Membership Service. Technical Report TR 94-37, University of Arizona, Tucson, AZ 85721, December 1994.

[18] L.E. Moser, Y. Amir, P.M. Melliar-Smith, and D.A. Argawal. Extended Virtual Synchrony. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 56–65, Poland, June 1994.

[19] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.

[20] D. Powell, editor. *Delta-4 - A Generic Architecture for Dependable Distributed Computing*. ESPRIT Research Reports. Springer Verlag, 1991.

[21] David K. Gifford. Weighted Voting for Replicated Data. In *Proceeding of the Symposium on Operating Systems Principles (SOSP)*, pages 150–163, 1979.

[22] Luís Rodrigues and Paulo Veríssimo. xAMp: a Multi-primitive Group Communications Service. In *11th Symposium on Reliable Distributed Systems*, pages 112–121, October 1992.

[23] Paulo Veríssimo and Luís Rodrigues. The NavTech Large-Scale Distributed Computing Platform. Technical Report RT-95, Broadcast Project, INESC, Rua Alves Redol 9-6^o, 1000 Lisboa, Portugal (in preparation).

[24] Pedro Antunes and Nuno Guimarães. Structuring Elements for Group Interation. In *Second Conference on Concurrent Engineering, Research and Applications (CE95)*, Washington D.C., August 1995.

[25] P. Antunes and N. Guimaraes. NGTool - exploring mechanisms of support to interaction. In *First CYTED-RITOS International Workshop on Groupware CRIWG '5*, Lisboa, Portugal, August 1995. CYTED-RITOS.