Tailoring Cooperation Support through Mediators

Anja Syri
GMD-FIT, German National Research Center for Information Technology
syri@gmd.de

Abstract: Cooperative processes are strongly influenced by their context. Individual and group experience, work practices, and the organisational setting are variable factors that help shape a cooperative work context. In an electronic environment the variability and dynamic nature of cooperative processes has to be taken into account. This suggests a requirement for configurability of basic cooperative functionality. In this paper, a concept for the development of a generic CSCW platform is proposed, which offers the possibility to configure its basic cooperation support functionality. Mediating objects are the key feature which enables the tailorability of functionality to the cooperative setting at run-time.

Introduction

Cooperative processes among human beings are very complex and strongly dependent on their environment; different organisational cultures as well as different teams develop individual forms of cooperation. Here distinctions can be made related to the structure of the cooperative process, the interdependence of the activities, or the transparency of the process. In order to develop a generic CSCW system, we need a framework offering basic cooperation support functionality without being limited to specific forms of cooperation. A suitable framework has to allow configuration on different levels—for the system designer, the system administrator as well as for the end-user—to adapt the system to the working environment.

J. Hughes et al. (eds.), Proceedings of the Fifth European Conference on Computer Supported Cooperative Work, 157-172.

We can distinguish two classes of CSCW systems which differ in the bandwidth of cooperation support they provide to their users. On the one hand, there are *CSCW applications* focusing on certain forms of cooperation. This group includes video and audio conferencing tools supporting communication, workflow systems focusing on coordination and shared workspace systems allowing the shared usage of information. On the other hand, efforts in developing *CSCW platforms* (e.g. LOTUS NOTES) have been undertaken, offering a cooperation environment with support for a range of cooperation activities.

Existing CSCW systems (CSCW applications as well as platforms) do not meet the requirements of generic cooperation support. Whereas applications are often closed, most platforms provide a programming interface and allow the integration of new tools. Nevertheless, these platforms neither support configuration of embedded basic cooperation support functionality (e.g. access control mechanisms, event services, or locking mechanisms) nor do they allow the integration of new cooperative functionality. Both features are necessary to adapt the system to a specific form of cooperation and the needs of the users.

This paper proposes a concept for the design of a lightweight CSCW platform. This platform provides a framework that allows the integration of basic cooperation support services by a developer or system administrator. Parts of this functionality are made visible to the users allowing them to become aware of how the chosen form of cooperation is supported by the system. The user has the possibility to do the fine-tuning: basic cooperation support can be complemented by additional services, for example services supporting norms that have been developed in the specific cooperation context.

The paper starts with the presentation of the requirements that a generic platform has to fulfil. The next section introduces our concept for the development of a generic CSCW platform and the configuration features it offers. This is followed by a presentation of related work and a comparison of the different approaches. The conclusion summarizes the results and explains the benefits of our concept.

Requirements

In developing a generic CSCW platform several aspects must be considered. This section presents requirements based on the experiences which we have made in the POLITeam project (Klöckner et al., 1995). The project aims to develop a CSCW platform that supports cooperative processes within large organisations. The development is done with special regard to the problems that are caused by the gradual movement of the German government, which will result in a distribution of labour between Bonn and Berlin. We follow a participatory design approach involving end-users from Ministerial sites (Mambrey et al., 1996).

The following list identifies some requirements for a generic CSCW platform.

Modelling the cooperative process. The basis of a CSCW platform is a model representing the entities of the cooperative process as well as the relationships among them. Much attention has to be paid to capture the dynamic aspects of the cooperative process within the model. How to represent activities, the progress of activities, the history? The model has to reflect rules and conventions that determine the cooperative process. Further issues that have to be considered here can be summarised by the keyword "awareness". Our users' demand for a visible cooperation process is described in (Sohlenkamp et al., 1997).

Supporting the transition between different forms of cooperation. Many CSCW applications have been developed, which focus either on well-structured or on less structured work processes. Whereas workflow systems impose a certain level of control on their users, shared workspace systems offer a high degree of freedom. Considering a concrete cooperative process, it is difficult to predict the "right" level of coordination support. A process that seems to be well-structured at the beginning and therefore best supported by a workflow system may require a broader scope of action for the persons involved at a later stage, which suggests the usage of a shared workspaces (and vice versa). Dourish and Bellotti (1992) show how the provision of awareness information can help members of a shared workspace to move smoothly between close and loose cooperation. Prinz and Syri (1996) describe a typical process in a department of a German Federal Ministry that underlines the need for an inter-linked usage of electronic circulation folders and shared workspaces. Learning from these experiences, a generic CSCW platform has to provide configurable cooperation support in order to support the dynamic character of real cooperative processes as well as the transition between different forms of cooperation.

Supporting context-specific norms. Cole and Nast-Cole (1992) stress the importance of group dynamics concepts for the development of CSCW systems. They identify the following stages in a group's development: "forming, storming, norming, performing, adjourning". Before cooperating effectively ("performing"), groups have developed a set of norms or group conventions. This shows the importance of supporting norms and conventions by a CSCW platform. One POLITeam user stated (Mark and Prinz, 1997):

> "There must be ground rules established for working together and dealing with groupware. The culture is lacking with the new technical possibilities to be able to get around this problem. The stronger the technical flexibility, the more rules must exist for how we can get around this."

Whether this results in just making visible these rules or forcing the user to keep to them depends on the cooperation context.

Tailorability of the system. A generic CSCW platform that can be applied to support various cooperative processes characterized by different forms of cooperation has to provide a broad range of basic cooperation support services. Since these services and their realization greatly affect the cooperative process they support, they have to be tailorable. Instead of hardcoding policies into the

services, they have to be realized in a modular way and kept separate from the services.

This list has introduced the most important factors that have to be taken into account when designing a generic CSCW platform. The next section introduces our concept for describing the design of a generic CSCW platform.

The Design Concept

The basis of a CSCW platform forms a data model representing the entities of a cooperative process (e.g. documents or folders) in the electronic system. We have decided in favour of an object-oriented model, because the object-oriented paradigm has several advantages:

- a natural way of modelling real entities and processes,
- encapsulation of objects and their functionality,
- modularization and
- reusability.

We do not want to describe a specific CSCW platform, nor do we want to focus on the concrete selection of classes. This section proposes a concept for the design of CSCW systems.

With regard to the generic character of the system—modelling different forms of cooperation and transitions among them—it is important to realize the cooperation support functionality in a modular way. This allows the adaptation of the functionality to the users' needs in two ways: first, the form of cooperation can be specified by selection of basic functionality. Second, configuration of this functionality should be possible to allow fine-tuning. Our concept proposes the following solution:

- The platform offers a broad range of cooperation support functions, each encapsulated in single service objects.
- The interaction of the basic service objects can be modified during run-time.
- Since cooperation processes strongly depend on their context (as the organisation or the team), the model allows specification of the cooperation support functionality to be applied dependent on the context in which the objects are used.

The service objects are called *CSCW enablers*. Enablers encapsulate basic cooperation support functionality and provide this functionality to objects. The primary purpose of CSCW enablers is to make objects "cooperation-aware".

All enablers belonging to an object are administered by a *CSCW mediator*. The mediator encapsulates the interaction among the enablers. It takes on the responsibilities of a mediator as described in (Gamma et al., 1995, pp. 273-282).

Figure 1 shows how a method call on a target object is forwarded to its mediator. The mediator knows which enablers belong to the target object and invokes the cooperation support functions offered by the enablers in addition to

the called method of the target. This concept is described in detail in the following sections.

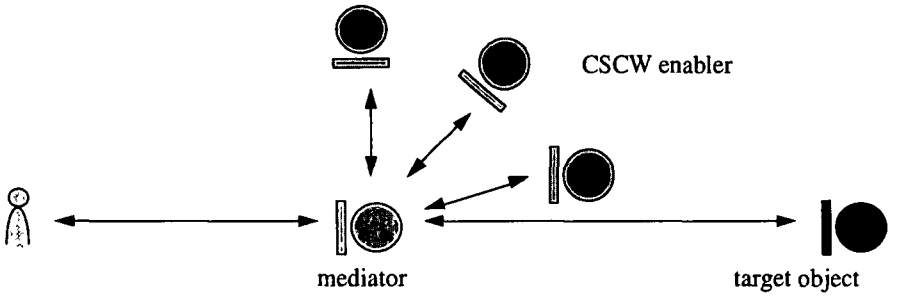

Figure 1: Dynamic enabling of a target object

CSCW Enablers

CSCW enablers encapsulate basic cooperation support functionality. They can modify the semantics of object methods on the target object as well as offer a different interface. The example of a container object that is enabled for use by several persons illustrates this. The methods of the container object can be enriched by an event mechanism which notifies changes to the users whenever a modifying method of the container object is called. In addition, the enriched object has to provide new methods, for example one to invite a person to take part in the sharing.

Ellis et al. (1991) distinguish three categories of cooperation support functionality: communication-enabling functionality, coordination-supporting functionality, and functions allowing the shared usage of common material. Based on this categorisation we want to identify basic functionality which can be encapsulated in CSCW enablers.

Communication

With regard to the required degree of synchronicity for communication, we can offer different communication enabling objects. Starting with an uncoupled communication between sender and receiver, functionality to exchange information on the basis of annotations or messages can be used. Enablers providing audio or video conferencing tools support a coupled form of cooperation.

The chosen form of communication is strongly dependent on two aspects. First, it requires that each cooperation partner has access to the technical equipment as well as the underlying infrastructure. The (technical) contactability has to be made visible in the system. Second, it depends on the social protocol, the style of cooperation that the group of cooperation partners has agreed upon. Just consider

a team member that wants to contact another to clarify a question: is it convenient to interrupt by using a synchronous communication tool or is sending a message the "right" way?

Coordination

Different forms of coordination can be applied in a cooperative context. Here too the supporting functionality is dependent on the character of the cooperative process to be supported. With regard to a well-structured process, it may be suitable that the system offers explicit coordination support. Different strategies are possible: coordination can be realized by a human coordinator or—if a technical solution is more appropriate—by a token mechanism. The contrary approach is to neglect coordination support in the CSCW model and to leave it to the social protocol. Prerequisite here is that the cooperation partners have the chance to become aware of what is going on. This approach is suitable for processes that are less structured.

Sharing

The shared usage of common information requires several basic cooperation support services. Compared to multi-user access, shareability is defined in (Mariani and Prinz, 1993) as multi-user access which is supported by two additional services: first, information on who is locking the object is kept by the system and made available to others trying to access the object (identified locking). Second, the users are informed about past and ongoing activities (awareness).

Let us assume that two persons of a cooperating team want to edit a document. After the first person (A) has opened the document, the second (B) tries to do the same. Different design alternatives are possible, some should be mentioned here:

- The access is denied and information about the current user of the object supplied.
- The access is denied, user A is informed about B trying to access the object, user B is informed that A is the current user.
- Both persons get the chance to negotiate how to proceed.

Dourish (1995) presents a different approach which is based on "divergence and synchronisation between multiple, parallel streams" instead of considering single streams of activity.

Classes of Enablers

Based on the categorisation of cooperation support functionality introduced in the last sections, we can identify different classes of CSCW enablers (cf Figure 2).

They:

- provide synchronous and asynchronous communication facilities to users,
- support coordination among cooperation partners by allowing specification of sequences of activities,
- control access to information objects,
- lock an object and inform persons trying to access the object about the current user,
- generate events to provide awareness information,
- allow users to specify interest in events and collecting events on their behalf.

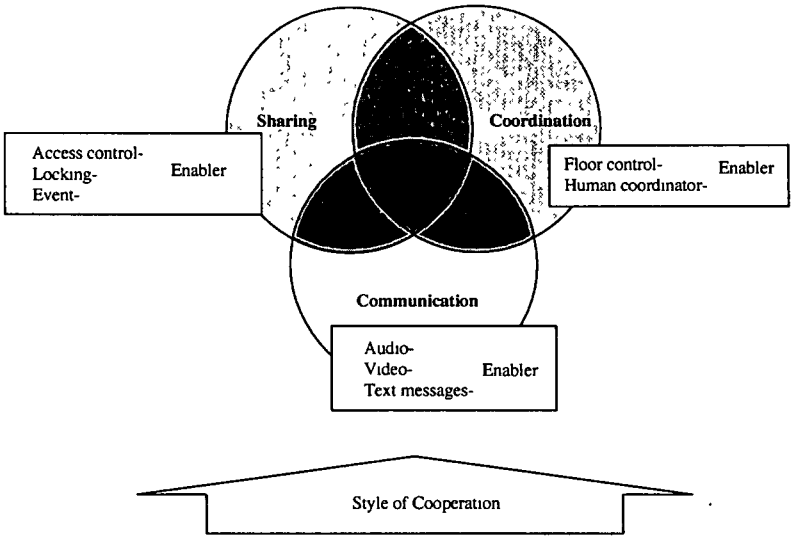

Figure 2. Different classes of enablers

Interface of a CSCW Enabler

In designing the interface of a CSCW enabler two aspects have to be taken into account: first, enablers are used to provide cooperation support to different kinds of objects. Second, the functionality provided by an enabler depends on which method is invoked on the target object.

For these reasons, all CSCW enablers offer a uniform interface (shown in Figure 3) that is independent from the interface of the enabled object. As will be shown in the next section, it is the task of the mediator to map object-specific methods onto methods provided by the enablers.

Each enabler method specifies how the corresponding method of the target object is extended. Additionally, the enabler interface provides the possibility to switch off this enabling functionality for single methods.

A closer look at the example of a shared workspace should illustrate this. Firstly, we have to clarify what we call a "shared workspace". A shared workspace offers its members a shared environment where they can store all documents they are working on. It is suitable to support less structured work processes, since it imposes no further processing rules on its members. Cooperation can be supported by providing awareness information to its members.

To realize this, an eventing enabler that informs the users about changes within the workspace is attached to the workspace object.

```
class eventingEnabler {
    delete (...);              //generates delete-event
    open (...);                //generates open-event
    openWithin (...),          //generates event indicating that an
                               //object within the workspace has
                               //been opened
    close ( ..);
    closeWithin (...);
    move (...);
    copy (...);
    share (...);
    add_object (...);
    remove_object (...),
    enable (method m);         //enabling the generation of
                               //specific events
    disable (method m);        //disabling the event generation
}
```

Figure 3: Interface of the eventing enabler

If a new object is added to the workspace, the mediator calls the corresponding method of the target object before invoking `add_object()` of the eventing enabler, which generates an event.

The users of the workspace may decide that they want to be informed when someone enters the workspace, but not when the person is leaving. In this case they disable the announcement of "leaving events" by using the method `disable(close)`. Calling `enable()` reverses this again.

So far we have identified some basic cooperation support services that can be used to add cooperation awareness to an object model, transforming it into a suitable basis for a CSCW platform. In order to model complex cooperative processes we have to support a mechanism that allows the combination of these enabling objects.

CSCW Mediator

A mediator administers all CSCW enablers belonging to one target object and encapsulates their interaction. For this reason it keeps a handle to the target object and to all its enablers. The mediator provides an interface for the registration of new enablers and the removal of registered enablers. These methods allow to keep

selection of enablers as dynamic as possible: CSCW enablers can be combined in an arbitrary way.

Initialisation

Whenever an object is created, a CSCW mediator for that object is instantiated and initialised according to a (default) configuration file. This allows specification of different policies by enumerating all CSCW enablers that are valid for the target object.

We differentiate between mandatory and optional CSCW enablers. Mandatory enablers are permanently assigned to an object, optional enablers can be removed from an object during run-time. This can be done under user control (cf Section "Configuration").

Figure 4 shows an example of a configuration file for an object that is enabled for the use as a shared workspace.

```
shared workspace :=   mandatory accessControl,
                      mandatory memberAdmin,
                      optional createShare,
                      optional communication,
                      optional eventing,
                      optional history.
```

Figure 4: Configuration file for a shared workspace

Invocation of a method

Whenever a method of an object is called, this call is first delegated to the mediator. The mediator calls the corresponding methods of the enablers that provide cooperation support functionality. Enablers invoked before the target object method is called provide pre-processing, whereas the remaining enablers allow post-processing.

Since the behaviour of an object is not only dependent on the object itself, but also on its context, the model provides a mechanism that takes this context into account. A method call to the target object first invokes the enablers attached to this target. If the target object is contained within an object, the mediator of this container object is called additionally and in turn invokes its enablers. This mechanism is applied recursively to further objects containing the container object.

Let us consider a simple example for illustration: if a document is deleted within a shared workspace, this has two effects: the document itself is deleted and it is removed from the workspace. According to this scenario, the invocation of the `delete()` method of the document should not only invoke the `delete()` method of all enablers attached to the document object, but furthermore invoke the `remove_object()` method of the enablers belonging to objects containing the

document object. Seen the other way round, this mechanism allows specification of a method for handling access to a container's contents.

To summarise, a mediator

- first calls the corresponding methods of the CSCW enablers attached to the object and its containers providing pre-processing,
- then invokes the object specific method on the target object (in case that a pre-processing enabler has denied access to the object, no further methods are invoked) and
- finally calls the corresponding methods of the CSCW enablers (attached to the object and its containers) providing post-processing.

Example

Based on a concrete example, we show how a single-user container object can be used as a shared workspace and how it can be configured to the users' need with the help of enablers and mediators.

Our experiences with the usage of shared workspaces have shown that some users prefer to edit documents within their personal working environment and to put them back in the workspace afterwards (Prinz and Syri, 1996). As a consequence, the document is not visible to other members during this time. The example for solution proposed here automatically creates a 'share' (which is a new access point, a handle to the document object) to the document when someone tries to remove it from the workspace. Both, the editor as well as the other members of the workspace, have access to the document during this time.

Compared to an object representing a simple container, a shared workspace object has to provide access control and to administer a list of members. Furthermore it can be enriched by a notification and history service, communication facilities, and functions supporting shared usage. An example for the configuration file was given in Figure 4. In order to simplify the example, we now concentrate on the invocation of enablers directly attached to the workspace and ignore the calls to enablers belonging to its container objects.

Each time a method of the container is called, the call is sent to its mediator. The pre-configured mediator delegates this call to its enablers. A closer look at the example of the removal of an object from the shared workspace illustrates this (cf Figure 5; the numbers indicate how methods are enabled).

Technically, the call to the `remove_object()` method is first delivered to the mediator. The mediator calls the access control enabler of the workspace object, which checks that the user is allowed to remove the document object. If this right is granted, the object can be removed by calling the corresponding method (`remove_from_container()`) of the target workspace object. Before this method of the workspace object is called, the create-a-share enabler creates a 'share' to the document and stores it in the workspace. Finally the event enabler method `remove_object()` is called and generates a notification.

Calls to the other methods are handled in a similar way. The methods `open()`, `close()` and `add_an_object()` are complemented by functions that control access to the document and send notifications about ongoing changes.

Compared to a single-user container, the shared workspace offers additionally methods like `invite_a_new_member()`, `exclude_a_member()`, `communicate()` or `get_history()`.

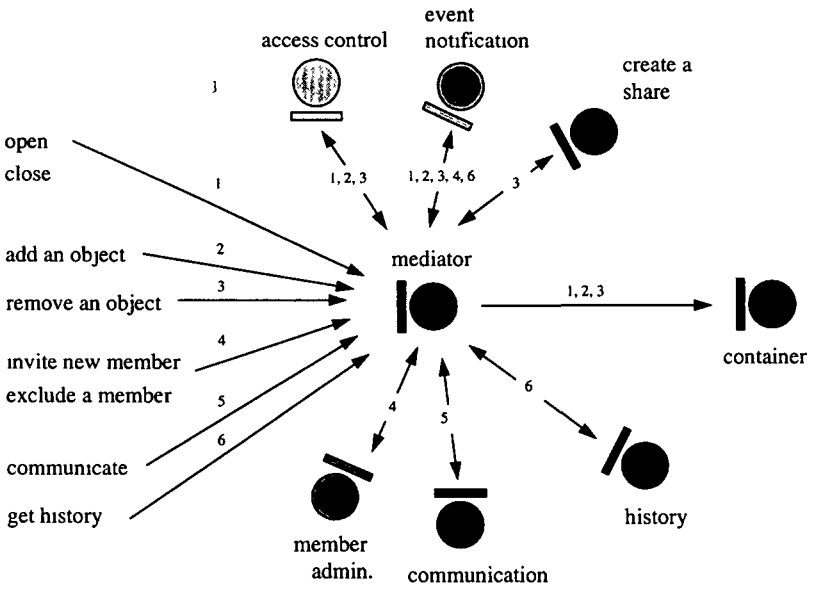

Figure 5. A container enabled for use as a shared workspace

This example shows how the combination of enablers allows the realization of a specific shared workspace behaviour. Having introduced the underlying concept, the next section shows how the basic cooperation support functionality can be adapted by configuration.

Configuration

Configuration of the cooperation support functionality is performed at different levels of abstraction: a default configuration can be specified in a configuration file, whereas a graphical user interface supports configuration during run-time.

The CSCW platform provides the underlying mediator pattern that supports the invocation of enablers. It is the responsibility of the designer of the platform to identify basic cooperation support functionality and to decompose this functionality into CSCW enablers.

The designer or system administrator can specify cooperation support functionality for an object or an object class through configuration files. In

addition to this technical solution, we provide a graphical user interface that gives end-users the possibility to adapt the behaviour of an object to its concrete working context.

First, the user can assign a new behaviour to the object. Here pre-defined sets of enablers are illustrated with the help of metaphors: if the user wants to share a single-user container object with further persons, the user can give it a "shared workspace behaviour". If it should be used in a strictly sequential manner, a "circulation folder behaviour" is appropriate.

Second, after the instantiation of the object itself and its mediator and enablers, the user can add CSCW enablers to and remove enablers from the object. For this reason, the enablers are made visible at the user interface. Figure 6 illustrates this: The tree on the left contains the complete list of available enablers. The enablers that are valid for the object "Workspace" (identified locking and a keep-privacy-enabler) are shown on the right side. The functionality of the first enabler has already been described above. When a user decides to move a document out of the shared workspace to work on it privately before putting the document back into the workspace, the user may not want to have every single step of modification be recorded in the history of the object. The keep-privacy enabler allows to specify the level of detail of information describing activities in the private working environment.

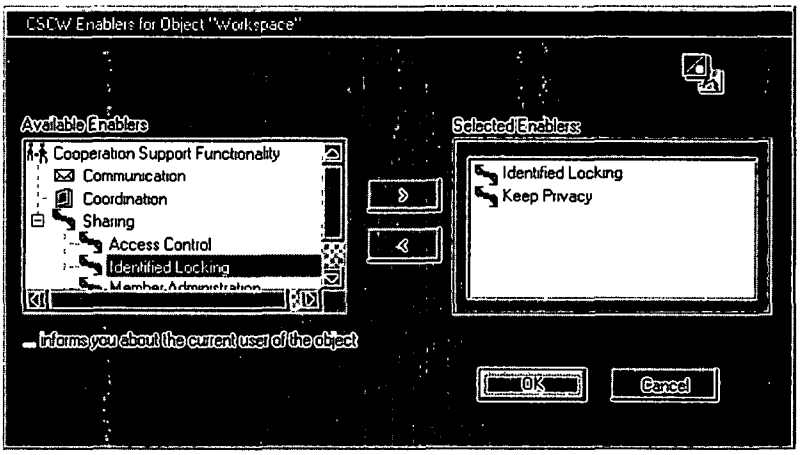

Figure 6: Specification of enablers for an object

To add more cooperative behaviour to the target object, the user moves the enablers from the left frame to the right. As a consequence, a corresponding enabler is initialised, attached to the object, and displayed within the right frame. To remove an optional CSCW enabler that is already assigned to the object, the user selects it on the right side and deletes it.

Third, the user can decide which activities are to be made cooperation-aware by enabling or disabling single methods of the CSCW enablers (the underlying methods were introduced in Figure 3). This is supported by context-sensitive menus attached to the enablers shown on the right.

Related Work

So far we have presented a concept for the design of a generic CSCW platform that allows the configuration of the cooperation support functionality. A lot of investigations have been undertaken with the aim to develop a CSCW platform, most of these developments concentrate on synchronous forms of cooperation. Representatives that can be mentioned here are RENDEZVOUS (Patterson et al., 1990), MMCONF (Crowley et. al, 1990), and GROUPKIT (Roseman and Greenberg, 1992). To portray these systems in simplified terms, the goal here is to provide the infrastructure for the distribution, initialisation and management of sessions.

Another group of CSCW-platforms is based on top of semi-structured messages or objects, as for example OVAL (Malone et al., 1992) or STRUDEL (Shepherd et al., 1990). The users interact by using semi-structured objects or messages; the structure of these entities is used to automatically invoke further functionality (such as the filtering of incoming messages or the modification of calendars). Configuration of the functionality is allowed by defining new message or object types, new structures and new rules. Nevertheless, a modular realization of cooperation support functionality is missing.

In (Navarro et al., 1993) the authors present a distributed environment, called MOCCA, for the design of open CSCW systems. This environment offers different services (e.g. a shared information space or an organisation information service) that can be used by CSCW applications realized on top of it. MOCCA differs from the approach presented here with regard to the level of abstraction: the first provides general services for CSCW applications, the latter allows tailorability of cooperation support functionality for single objects.

PROSPERO (Dourish, 1996) provides a framework for the development of CSCW systems. This development is supported by allowing the integration of application functionality into the framework itself. Whereas PROSPERO focuses on the development of CSCW systems, our concept concentrates on the configuration of the developed system during run-time.

COLA (Trevor et al., 1993) is an object-oriented platform supporting cooperative processes. It provides fundamental but basic mechanisms for the development of CSCW applications. The system uses a concept of object adapters (Trevor et al., 1994) to provide cooperation support for objects. Compared to COLA, the approach presented here focuses on a different level of abstraction,

considering how the cooperative functionality should be provided, how it can be made visible to and tailorable by the users.

A different approach is presented by Bentley et al. (1992). The authors propose an architecture that allows the tailoring of multi-user displays by system designers in conjunction with users. Shared interaction objects, called user display agents, handle details about the interaction with and presentation of the shared information store. Thus the role of the user display agents can be compared to the role of mediators introduced in this work, but whereas the first deal with presentation issues, the latter focus on underlying cooperation support functionality.

Conclusion

In this paper we have presented an object-oriented concept for the design of a generic CSCW platform. This concept supports the modelling of cooperative processes by

- encapsulating cooperation support functionality into separate objects, called CSCW enablers,
- encapsulating the interaction of these enablers into a CSCW mediator.

Instead of hard-coding cooperation support into the object model, the concept allows to dynamically add or remove functionality by using enablers and mediators. This allows to flexibly combine basic enabler functionality in order to realize complex cooperation support functionality. Enabled single-user objects can show a new behaviour and be shared among users.

Configuration of this functionality can be done on several levels: first by selecting appropriate CSCW enablers, second by specifying their interaction and the policies employed. This configuration allows support of different forms of cooperation and the transition between different forms (by exchanging single enablers) as well as to represent context-specific norms (by adding further enablers).

There are still open questions that have to be examined concerning the configuration features provided by the concept. Different cooperation situations have to be looked at in order to identify typical configurations of enablers. The employment of metaphors that was briefly mentioned can provide support to make the configuration easier for the user.

Besides developing a CSCW platform from scratch, the concept allows to provide additional functionality to existing CSCW platforms as well. Prerequisite here is that the platform offers a programming interface. Two different approaches are possible: the modification of the underlying object model of the platform or—if this is not possible—the development of an external application which realizes enablers and mediators and makes use of the existing functionality of the platform via the programming interface.

This concept is implemented within the framework of the POLITeam project. However, we do not want our users to design their own cooperation environment, but to do the "fine-tuning". Having stressed the demand for the support of group conventions within a CSCW system, we expect that our users experiment with CSCW enablers representing conventions. The example of removing a document out of a shared workspace was mentioned before. How can this problem be solved? Should this removal be allowed or forbidden? If it is allowed, should the user be informed about the consequences of this action? Or should the decision be left to the user, but recorded? We are looking forward to how our users will use and evaluate this approach.

Acknowledgements

Thanks to Ludwin Fuchs, Wolfgang Prinz, and Markus Sohlenkamp for valuable discussions on many topics raised here. Thanks also to Richard Bentley and Gloria Mark for their constructive comments on earlier versions of this paper.

Bibliography

Bentley, R., T. Rodden, P. Sawyer, and I. Sommerville (1992): "An Architecture for Tailoring Cooperative Multi-User Displays", CSCW 92, Toronto, Canada, in. *Proceedings of the Conference on Computer-Supported Cooperative Work*, ACM Press, pp. 187-194.

Cole, P. and J. Nast-Cole (1992) "A Primer on Group Dynamics for Groupware Developers", in: *Groupware*, Marca, D. and G. Bock (eds.), Los Alamitos, California: IEEE Computer Society Press, pp. 44-57.

Crowley, T., P. Milazzo, E. Baker, H. Forsdick, and R. Tomlinson (1990): "MMConf: An Infrastructure for Building Shared Multimedia Applications", CSCW 90, Los Angeles, CA, in: *Proceedings of the Conference on Computer-Supported Cooperative Work*, ACM, pp. 329-342

Dourish, P. (1995): "The Parting of the Ways: Divergence, Data Management and Collaborative Work", ECSCW 95, Stockholm, Sweden, in: *Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work*, Kluwer Academic Publishers, Marmolin, H., Y. Sundblad, and K. Schmidt (eds.), pp. 215-230

Dourish, P. (1996): "Consistency Guarantees: Exploiting Application Semantics for Consistency Management in a Collaboration Toolkit", CSCW 96, Boston, MA, in: *Proceedings of the Conference on Computer-Supported Cooperative Work*, ACM Press, Ackerman, M S. (ed.), pp. 268-277.

Dourish, P. and V. Bellotti (1992): "Awareness and Coordination in Shared Workspaces", CSCW 92, Toronto, Canada, in: *Proceedings of the Conference on Computer-Supported Cooperative Work*, ACM Press, pp. 107-114.

Ellis, C., S. Gibbs, and G. Rein (1991). "Groupware: Some Issues and Experiences", *Communications of the ACM*, Vol. 34 (1991) 1, pp. 38-58.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995): *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series; Addison-Wesley Publishing Company.

Klöckner, K., P. Mambrey, M. Sohlenkamp, W. Prinz, L. Fuchs, S. Kolvenbach, U. Pankoke-Babatz, and A. Syri (1995): "POLITeam Bridging the Gap between Bonn and Berlin for and with the Users", ECSCW 95, Stockholm, Sweden, in: *Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work*, Kluwer Academic Publishers, Marmolin, H., Y. Sundblad, and K. Schmidt (eds.), pp. 17-31.

Malone, T. W., K.-Y. Lai, and C. Fry (1992): "Experiments with Oval: A Radically Tailorable Tool for Cooperative Work", CSCW 92, Toronto, Canada, in: *Proceedings of the Conference on Computer-Supported Cooperative Work*, ACM Press, pp. 289-297.

Mambrey, P., G. Mark, and U. Pankoke-Babatz (1996): "Integrating User Advocacy into Participatory Design: The Designers' Perspective", PDC 96, Cambridge, MA, in: *Proceedings of the Participatory Design Conference*, Blomberg, J., F. Kensing, and E. Dykstra-Erickson (eds.), pp. 251-259.

Mariani, J. A. and W. Prinz (1993): "From Multi-User to Shared Object Systems: Awareness about Co-Workers in Cooperation Support Object Databases", in: *Informatik, Wirtschaft, Gesellschaft*, Reichel, H. (ed.), Berlin. Springer-Verlag, pp. 476-481.

Mark, G. and W. Prinz (1997): "What happened to our Document in the Shared Workspace? The Need for Groupware Conventions", INTERACT 97, Sydney, Australia, in: *Proceedings of the Sixth IFIP Conference on Human-Computer Interaction*, Chapman & Hall, to appear.

Navarro, L., W. Prinz, and T. Rodden (1993): "CSCW requires open systems", Computer Communications, Vol. 16, No. 5, pp. 288-297.

Patterson, J. F., R. D. Hill, S. L. Rohall, and W. S. Meeks (1990). "Rendezvous: An Architecture for Synchronous Multi-User Applications", CSCW 90, Los Angeles, CA, in: *Proceedings of the Conference on Computer-Supported Cooperative Work*, ACM, pp. 317-328.

Prinz, W. and A. Syri (1996): "Two Complementary Tools for the Cooperation in a Ministerial Environment", PAKM '96, Basel, Switzerland, in: *Proceedings of the First International Conference on Practical Aspects of Knowledge Management*, Workshop "Knowledge Media for Improving Organisational Expertise", Wolf, M. and U. Reimer (eds.).

Roseman, M. and S. Greenberg (1992): "GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications", CSCW 92, Toronto, Canada, in: *Proceedings of the Conference on Computer-Supported Cooperative Work*, ACM Press, pp. 43-50.

Shepherd, A., N. Mayer, and A. Kuchinsky (1990): "Strudel - An Extensible Electronic Conversation Toolkit", CSCW 90, Los Angeles, CA, in: *Proceedings of the Conference on Computer-Supported Cooperative Work*, ACM, pp. 93-104.

Sohlenkamp, M., L. Fuchs, and A. Genau (1997): "Awareness and Cooperative Work - The POLITeam Approach", HICCS 97, Wailea, HI, in: *Proceedings of the Hawaii International Conference on System Sciences HICSS-30*, IEEE Computer Society Press, Nunamaker, J. F. and R. H. Sprague (eds.), vol. 2, pp. 549-558.

Trevor, J., T. Rodden, and G. Blair (1993): "COLA: a Lightweight Platform for CSCW", ECSCW 93, Milano, Italy, in: *Proceedings of the Third European Conference on Computer Supported Cooperative Work*, Kluwer Academic Publishers, Michelis, G. D., C. Simone, and K. Schmidt (eds.), pp. 15-30.

Trevor, J., T. Rodden, and J. Mariani (1994): "The Use of Adapters to Support Cooperative Sharing", CSCW 94, Chapel Hill, NC, USA, in: *Proceedings of the Conference on Computer Supported Cooperative Work*, ACM/SIGCHI, Furuta, R. and C. Neuwirth (eds.), pp. 219-230.