Innovative Concepts for Configuring Shared Workspaces through Visual Programming

Marita Duecker, Wolfgang Mueller, Jessica Rubart
C-LAB
Fuerstenallee 11
33102 Paderborn
Germany
{mary, wolfgang, jessica}@c-lab.de

Abstract

CSCW systems like BSCW (Basic Support for Cooperative Work), HyperNews, and Lotus Notes have been successfully introduced to support the cooperation of geographically distributed work groups. Unfortunately, some of these software systems are not flexible enough to be customized easily to the requirements of an individual user. Most of the environments can only be configured by experts through complex programming.

In this article we introduce VIPspace, a shared workspace in the sense of exchanging and processing shared objects between members of distributed work groups. VIPspace can be easily adapted to the user's individual views and needs by means of a visual programming language. In VIPspace, objects of the shared workspace are moved by drag & drop between task fields. Task fields are programmed by visual 'if-then' rules applying a combination of icon- and form-based techniques. When a document represented by an icon is dropped on a task field its rules are checked and actions of the selected rules are executed. VIPspace is finally demonstrated in an Internet course application.

1. Introduction

With the wide availability of global computer networks efficient cooperative computer-based applications are emerging. A wide range of CSCW (Computer Supported Cooperative Work) systems with videoconferencing, electronic mail, discussion forums, etc. are already part of our daily business.

Services can be distinguished with respect to short/long distances as well as synchronous/asynchronous communication [7, 3]. Applications for asynchronous communication such as electronic mail and newsgroups are extremely helpful to coordinate communication and cooperation in distributed work groups, i.e., for people which are distributed over various locations, timezones, and groups which need general off-line information management.

Shared workspaces are groupware environments which combine those applications to a complete system. They typically provide a framework for a joint view on shared information for participants in different locations to interact with [5]. Shared workspaces typically integrate electronic mail, videoconferencing, shared electronic whiteboards, and a joint view to shared digital objects like documents, voice, pictures, and movies. In contrast to classical email their joint view typically provides a synchronous well-structured immediate access to private and shared objects while asynchronous email is received as an unstructured linear stream of digital information. Emails are structured in private folders after receipt and typically published by explicitly sending them.

Popular examples of shared workspaces are BSCW (Basic Support for Cooperative Work) [4], HyperNews [8], Lotus Notes [13], Livelink 7 [2], and Microsoft Exchange [1]. Present versions of these systems are not flexible enough to be easily adapted to the requirements of individual applications and users. They only support a limited set of predefined different views on stored objects with very limited processing facilities.

We introduce the visual shared workspace VIPspace (VIsually Programmable Workspace). VIPspace is adaptable to individual user's needs through a combination of a form- and icon-based visual programming language [11]. In VIPspace the user's desktop is divided into a private and a (shared) global workspace. The user can create task fields for structuring and processing individual digital objects like documents on both of these areas. Objects can be easily moved as icons by drag & drop between private and global task fields. Each task field can be programmed by specifying a set of rules. Rules compute object attributes and execute simple actions like spawning tools, copying/moving/deleting documents. Rules are executed on an object as soon as the object is dropped on a task field. Rules are programmed through a combination of a form- and icon-based visual interface. By the means of rules the user has the ability to define a sequence of simple document processing activities for each task field. By those means the

user can easily reconfigure tasks by a few mouse clicks. The work described herein mainly focuses on VIPspace's novel visual concepts rather than on the underlying services of a shared workspace.

The remainder of this article is structured as follows. Next section discusses related work. Thereafter, we introduce VIPspace. Section 4 gives a short VIPspace example of a simple Internet course. After presenting our preliminary user studies we outline our implementation. The final section closes this paper with a conclusion.

2. Related Work

Currently available shared workspaces typically integrate facilities for very domain-specific applications and are in most cases tightly coupled with the underlying implementation. They quite often focus on the secure and reliable document exchange rather than on advanced user interaction with comfortable access to all facilities of the workspace. For most of these systems we can identify drawbacks in

- configuration/reconfiguration of individual views and processes
- general manipulation of documents, i.e., save, copy, move from/to shared workspaces and publishing to shared workspaces

In particular, the second point is surprising since the listed functions are typical ones which are most frequently applied during the user's interaction with the system.

HyperNews [8] basically provides advanced newsgroup facilities with notification services for distributed work groups. The user interface gives a tree view on shared information collected in hierarchical folders. It is not possible for a user to customize look & feel or structure of the workspace to specific requirements. For additional services HyperNews provides a built-in facility for notification by email. Users add messages by filling a form after clicking a button. Copying and moving of messages in administration mode requires the specification of the destination path. For deleting a message the user simply has to click a button after selecting the message.

BSCW (Basic Support for Cooperative Work) [4] extends standard WWW servers by shared workspace facilities for document and message exchange. These objects are structured in hierarchical folders. MIME types of objects are indicated by different icons which can be individually customized to a specific application. BSCW can be additionally configured/reconfigured during runtime through its Python interface. More advanced workflows can be added through WebFlow [6] by workflow diagrams. The up- and downloading of objects is supported through file selection boxes. Additional helpers support the uploading of objects via drag & drop. Deletion of objects requires to check the objects' box and executing 'delete'. Moving an object requires a set of activities since after selecting the object and clicking on 'cut' the object is moved to the private bag. Therein, the object has to be selected again. 'Drop' finally pastes it to the final destination.

Lotus Notes [13] provides structured views on objects in shared databases. Objects can be accessed via forms which can be individually composed by each user. Additional functionality can be added by programming agents in a textual scripting language. Due to the individual application the scripting language requires more or less advanced programming skills. Due to their individual context objects can be easily deleted, copied, and moved by drag & drop or by cut & paste.

Open Text Livelink 7 [2] implements a shared repository for document management organized in hierarchical folders. Document categories can be defined through a set of 'custom' attributes. The general user interface cannot be customized to a specific application. Livelink 7 does not support general facilities for processing documents. However this is supported in a separate component, i.e., Livelink Workflow, which allows the management of general workflows. Deletion of documents can be performed through an info page. Adding and moving documents requires a couple of activities. After navigating two forms the user has to enter the full path of a document and an additional name before clicking a button to complete the action.

Microsoft **Exchange**'s [1] server manages directories and the exchange of shared objects. Objects can be accessed through an Exchange client. The client gives a common tree view to shared and private objects. Objects can be easily moved by drag & drop from private to public information space. The Exchange client does not support an individual reconfiguration of the graphical view. A set of built-in activities (alert, delete, move, copy, forward) can be triggered by rules when a message arrives. The condition of a rule checks for textual patterns in sender, receiver, subject, and content of the received message. The patterns have to be specified in given text fields. Conditions can be combined by logical OR and AND.

Considering the above approaches only Lotus Notes and partly BSCW permit an individual customization of the view on visually structured documents. However, Notes only supports the generation of different views through combining forms. For advanced document processing, all of the above PC-based tools can be partly extended through the programming of their COM interface. In Lotus Notes the user has to specify additional actions in a scripting language. BSCW gives a high extensibility through Python, but it requires advanced programming knowledge similar to Lotus Notes. Though Microsoft's Exchange basically provides rule-based specification it is limited to email filtering. With the exception of Microsoft's Exchange and partly BSCW all of the above environments require a quite complex sequence of activities for moving/copying documents between the private and shared workspace. The user has to perform time consuming navigation through a couple of forms clicking several buttons.

VIPspace addresses all these drawbacks. VIPspace gives high flexibility in individual visual customization combined with necessary simplicity with respect to task specific requirements and users' preferences. In VIPspace, even a user with no programming skills can easily customize his/her individual view through a form-/icon-based visual programming language. Inspired by Repenning's Agentsheets [9] this language gives immediate access to VIPspace's complete functionality. Documents can be easily moved and copied between the private and global workspace. Unlike other systems, VIPspace requires a single drag & drop rather navigating through several forms clicking a couple of buttons. By the means of rules the user actually has the ability to define 'micro' workflows. In contrast to existing workflow systems the user still keeps complete control and overview of the individual task since documents are explicitly moved between tasks.

3. VIPspace

VIPspace is a shared workspace which can be individually customized by the means of a form-/icon-based visual programming language. The user can easily adapt the configuration of the workspace at any time to his/her current needs and preferences. After outlining VIPspace's basic facilities as introducing the role of its individual objects and components like task fields, we outline their programming by defining rules, and application.

3.1. Workspace

The VIPspace workspace is initialized by the workspace administrator. The administrator creates an initial configuration of task fields on the global as well as on the local workspace. The administrator specifies the list of users, their passwords, and their access rights. The administrator assigns advanced access to users which shall have complete control over all documents and which have administration access to the programming of rules and creation of global task fields.

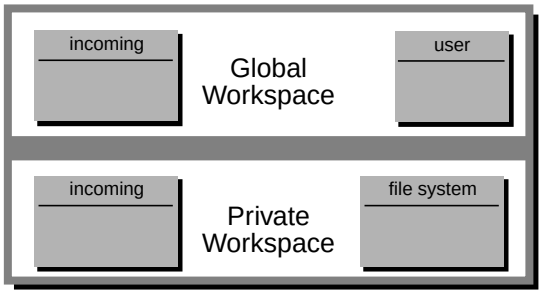

Figure 1. Initial VIPspace Desktop.

When starting VIPspace a user is asked to enter user id and password. After logon, the VIPspace desktop comes up with the global workspace in the upper and the private workspace in the lower section of the desktop. Workspace documents are represented by different icons which can be moved between task fields. Task fields are given as subareas on the desktop. The behavior of a task field is specified by a set of rules. The condition of each rule checks values of document attributes. When dropping an icon on a task field the first rule with a true condition is selected and the list of its actions is executed in sequential order. Task fields are either located in the private or in the global workspace. Private and global workspace are separated by a horizontal bar as it is sketched in Figure 1. Figure 2 depicts the general logical structure of VIPspace.

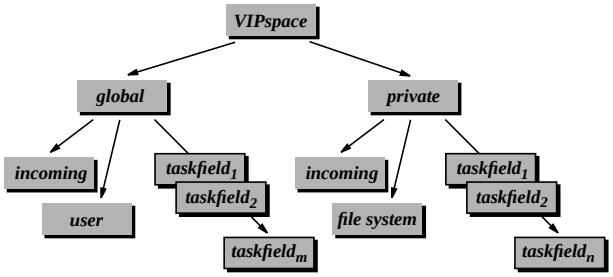

Figure 2. VIPspace Components.

The **global workspace** gives a joint view and access to the objects shared by all users. In the global workspace each user has the same view to the task fields and their objects. Task fields can only be configured and reconfigured, i.e., programmed, by the workspace administrator or users with administratior access. Accessing objects as well as task fields in the global workspace for reading and writing requires read and write permission.

When an object is moved from the local to the global workspace the individual user specifies the document's access permissions. Read permission means that the object is not confidential and can be read by all users. Write permission gives other users the ability to modify and delete the object. Normal users can only drop objects on task fields with write permission and move objects from task fields with read permission.

An user can either directly move an object from the private workspace to a global task field by drag & drop or the object can be submitted by an executed rule which sends it to 'global'. In the latter case the object appears in the *incoming* task field of the global workspace. In order to automatically distribute received documents to other task fields, the *incoming* task field can be programmed by a set of rules. So, incoming objects can, for instance, automatically be sorted by author, name etc. An additional predefined *user* task field displays icons of users which are currently logged on.

The **private workspace** gives users the ability to individually configure his/her private administration of objects. Whereas objects in the global workspace are visible to and accessible for all users, the private workspace is completely individual to each user. Like the global area it

has a predefined task field *incoming* for receiving objects via the VIPspace's communication system. Additionally, the private space has an extra field *file system* for integrated access to files on the local file system. Files can be movmoved by drag & drop from this field. The user can create an arbitrary number of private task fields. Private objects can be easily published from the private workspace by simply dragging them over the horizontal bar and dropping them in the upper global area.

3.2. Objects

Each electronic object (e.g., document, message) is represented by an icon located in the area of a task field. Each object has a (semantic) type. The representation of the object, i.e., the icon, is associated with that type. A newly uploaded object gets a default icon since the attribute of the semantic type is initially not specified. The icon changes after the type is manually assigned by the user. The type of the object typically changes through the life time of the object which modifies the corresponding icon. Figure 3 gives the example of 5 different representations of a document object. VIPspace provides a set of default icons given as gif files whose identifiers correspond to type identifiers.

Figure 3. Different Icons for Different Types.

An object is completely specified by its attributes. We distinguish between predefined and user-defined attributes.

Predefined attributes are attributes which are initialized and partly automatically managed by the system. VIPspace currently includes the following predefined attributes

- *semanticType*, (STRING). Defines a string indicating the objects' intended use. Different icons can be associated with different semantic types.
- *mimeType*, (STRING). Holds the MIME (Multipurpose Internet Mail Extension) type of an object. The attribute is evaluated when launching an associated viewer, e.g., when executing an action.
- *author*, (STRING). Includes the name of the user who created the object.
- *name*, (STRING). Holds the object's name. This is typically the file name when being copied from the file system.
- *modification*, (DATE). Gives the date and time of the last modification.

User-defined attributes. An arbitrary number of additional attributes can be added to any object. They strongly depend on the application domain. In its current version VIPspace supports attributes of type INTEGER, STRING, and other Java standard data types. Examples of user-defined attributes are

- *expirationDate*, (DATE). The date where the processing of the object is expected to be finally completed.
- *receiver*, (STRING). The email address of final receiver of the object.

Figure 4 shows a schematic example of a document and its icon with five predefined attributes and one user-defined attribute (myAttribute).

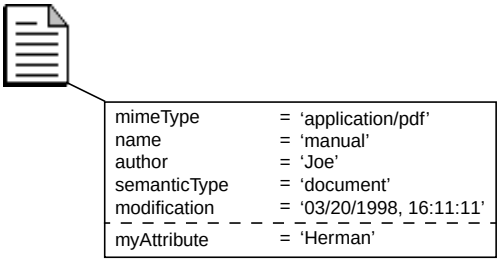

Figure 4. Attributes of a Document.

3.3. Task Fields

Task fields are subareas of the global or private workspace. For a newly configured VIPspace the administrator typically provides an initial set of application specific task fields. Task fields hold and process objects. When creating a task field the user first has to enter the name of the task field. Thereafter, he/she places it somewhere on the workspace. Once created the user can move objects in form of icons to that task field. Task fields in the global workspace have read and write permissions as already outlined before. Figure 5 shows two task fields with their names in the upper section.

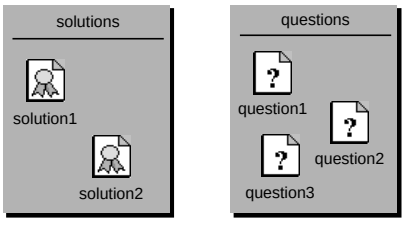

Figure 5. Two Task Fields.

A newly created task field initially has no behavior. It just keeps objects which are moved to that field. A task field can be programmed by the means of visual if-then rules. When double clicking on its area a rule editor is started which gives the user the ability to specify rules. When dropping an object on a field the system checks the condition of each rule in sequential order. If the condition

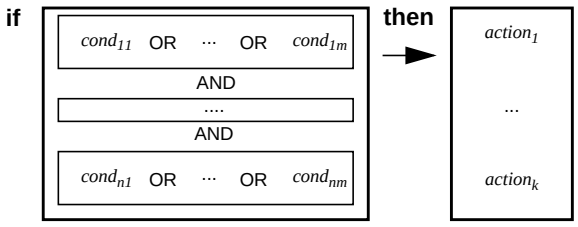

Figure 6. Rule of a Task Field.

of a rule matches the values of the dropped object's attributes, the corresponding actions are executed in sequential order. A rule is a tuple with a set of conditions and a list of actions (see Figure 6). A condition is a conjunction of OR-related basic conditions. A basic condition $cond_{xy}$ with $x \in \{1, \ldots, n\}$, $y \in \{1, \ldots, m\}$, $n, m \in \mathbb{N}$ has the form

a op v,

where a is a single element of the set of attributes A, v is a value in the range of a, and *op* is an operator for a. For integer this operator is in {=, <, >, ==, !=} and for string it is in {!=, ==, strlength}. Conditions are specified by dragging an (condition type) icon, dropping it on the condition field of the rule, and specifying its values in a form. Table 1 shows icons of predefined attributes. When clicking on one of those icons a form opens which has to be filled.

Attribute	Icon
semanticType	
mimeType	
author	
name	
modification	

Table 1. Predefined Attributes.

For actions we can distinguish local and global actions.

Local actions are actions which do not move or copy documents to another field or workspace. Table 2 gives examples of some local actions.

Gobal actions move or copy the document to/from the current task field, such as up/downloading it to/from VIP-space, copying or moving it to another task field in the private/global workspace, and sending messages or emails.

Table 3 gives some examples of global actions and their iconic representation. The current system provides a set of 15 actions. However, this set can be easily extended for other applications since the corresponding forms have a very generic structure for a semi-automatic creation.

Action	Icon
modifyAttributes	
launchApplication	
talkToUser	
deleteDocument	

Table 2. Local Actions.

We briefly demonstrate the application of rules by a simple example which implements a sorter for documents (see Figure 7). For this example, consider two (semantic) types of documents: '*information*' and '*solution*'.

Action	Icon
moveTo-,copyToGlobal	,
moveTo-,copyToPrivate	,
moveTo-,copyToTaskfield	,
moveTo-, copyToLocalFileSystem	,
sendMessageToUser	
emailToUser	

Table 3. Global Actions.

The implementation of a task field which automatically shifts documents of specified types to the task fields '*info*' and '*solutions*' requires two rules:

if (*semanticType == 'information'*)
then *moveToTaskfield(info);*

if *(semanticType == 'solution')*
then *moveToTaskfield(solutions);*

moveToTaskfield is a predefined action which sends the documents to the task field which is specified as a parameter of that function.

Whereas the previous section has introduced the basic concepts of the individual VIPspace components the next section presents a complete VIPspace example.

4. Example

Our example is for information management support in the context of a university course which is partially held via Internet. Today, course materials, worksheets, and other information sources are mostly exchanged in many different formats through emails, HTML-documents, and so on. This unstructured information exchange causes a lot of overhead for students and lecturer due to little tool support for efficiently handling the contents of emails.

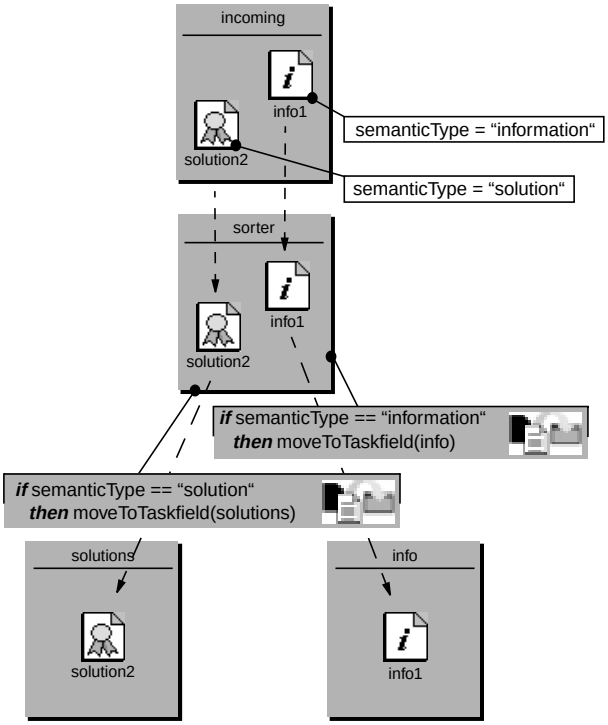

Figure 7. Sorting Documents.

The example introduces how VIPspace helps the management of such a course. In order to keep the scenario short special emphasis is given to the lecturer although every participant of such a course can benefit from the usage of VIPspace. The lecturer can adapt VIPspace to individual and course-related needs. We focus on activities where

- the lecturer distributes a worksheet among students.
- students submit their solutions back to the lecturer.
- after grading the submitted solutions, the lecturer makes all solutions with a grade better or equal B available to the students. All information about the author is removed.

The lecturer's individual VIPspace view is given in Figure 14. This figure shows the lecturer's private workspace at the bottom and the global one which is also visible for all students on the top. Both areas are separated by a grey bar.

In this scenario, we first outline the representation of objects before discussing the organization of the private and the global workspace.

4.1. Objects

The set of document objects (worksheet, submission, exemplary solution, question, information) required for our example is given in Figure 4. Figure 14 gives a configuration of their application in the context of this example. After the elaboration by a student the submission is displayed as a checked document icon. The returned and anonymized documents come in form of documents with a decoration. When selecting 'open object' with the right mouse button on the checked icon, for instance, a form with the individual values is opened as shown in Figure 8.

Figure 8. Document Attributes.

4.2. Workspace

The lecturer's private workspace (lower section of Figure 14) has the predefined task fields *incoming* and *local file system*. The *incoming* task field holds all received messages and documents. We define two new task fields in addition: *submissions* and *questions*. The lecturer uses the first task field to drop incoming *submissions* on. Incoming *questions* are dropped on the second field. We briefly demonstrate VIPspace programming by the creation of the task field *submissions* and specifying its behavior.

For the *submissions* task field we implement the behavior as it is depicted in Figure 9. When dropping an A- or B-graded document of (semantic) type *submission* on that field two actions are executed which

- modify the document's attributes: the grade is deleted; type and author are modified to *solution* and *anonymous*, respectively.
- the so modified document is automatically moved to the task field *solutions* in the global workspace which finally holds the stripped A- and B-graded solutions.

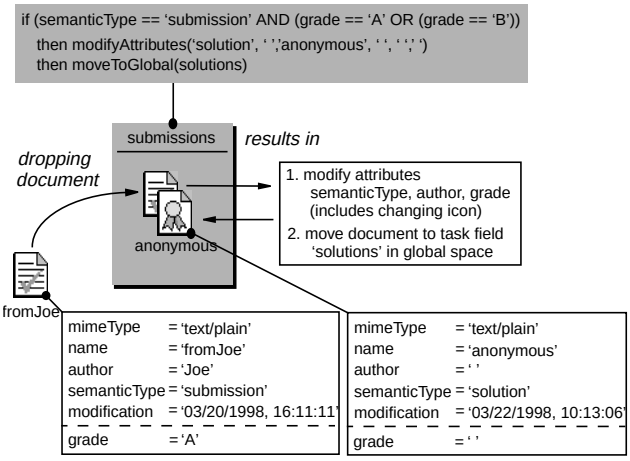

Figure 9. Modifying Documents.

The new *submissions* task field is created through the pop up menu which opens with the right mouse button on the private workspace. After entering the name 'submissions' the user places the task field on the private workspace. The task field is programmed by a double click on that field which opens the rule editor. The rule editor has four subareas (see Figure 10). On the leftmost area the user sees a list of available conditions. There is typically one condition for each attribute of a document. The rightmost area holds the set of available actions. The two areas in the middle define the actual rule with the conditions on the left and the list of actions on the right. A rule is specified by dragging a condition/action icon and dropping it on the area of the actual conditions/actions. In the area of actual conditions we specify a conjunction of disjunctions. Each row holds the set of conditions which are combined by AND. OR-ed conditions of one row are enclosed by a solid box.

In the example of Figure 10 we see from the top to the bottom the iconic representation of conditions for

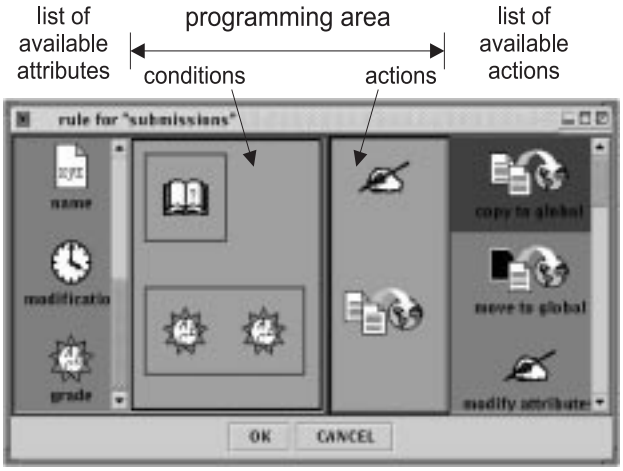

Figure 10. Rule Editor for One Rule.

if (semanticType == 'submission')
AND
((grade == 'A') OR (grade == 'B'))

Individual values of conditions are specified or modified when double clicking on them. Thereafter, a form opens which allows the modification of values in text fields or by selecting a button.

Figure 11 shows the form for the first condition which refers to the document's attribute *semanticType*. Herein, the user can select different available string operations (*equals*, *equals not*) and a type. In the given figure *submission* is currently selected. When a document is dropped on that task field and the set of conditions of the above rule evaluates to true the sequence of actions on the right is executed from the top to the bottom. In this example we have specified two actions. The first action modifies the document's attributes. The second action copies the document to a task field of the global workspace.

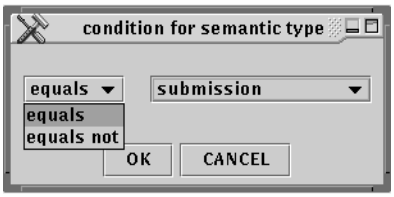

Figure 11. Conditions for Semantic Type.

Actions are specified similar to conditions, i.e., double clicking on the individual icon opens a form. When clicking on the first action *modify attributes* in Figure 10, for instance, the form of Figure 12 opens. In this form the user can determine how attributes of the document are modified. In our example we strip the author's name by exchanging it to 'anonymous'. The value of *semanticType* is changed to 'solution' and the value of *grade* is removed. The values of other text fields are kept unchanged (no change) which modifies the document by only changing its *semanticType*, *author*, and *grade*. Thereafter, the subsequent action *copyToGlobal* moves the document to the task field *solutions* in the global workspace.

For implementing basic functions of the course we can additionally create three other task fields in the global workspace. These are *worksheets*, *solutions*, and *information*. The latter one as well as the predefined task field *incoming* have been iconified in Figure 14. The *user area* shows icons of all users who are currently active in VIP-space.

solutions is a task field with one simple rule sending a notification to all students when an exemplary solution is received. Task field *worksheets* has a similar behavior. It holds the set of worksheets and sends an email when a new worksheet is received. Recall here, that the student's view of the global space is the same as the lecturer's one.

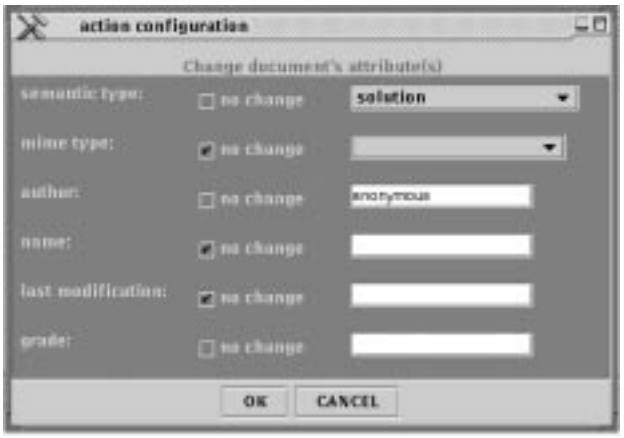

Figure 12. Changing Attributes.

5. User Evaluation

We have performed basic informal empirical evaluations of the "Internet Course" based on the think aloud method [12] where users are encouraged to report their individual actions when carrying out task scenarios on working prototypes or implemented systems. Think aloud protocols include a problem report and recommendations for product improvements. This method already generates meaningful results with 5 to 10 test subjects.

We have performed our evaluation with 7 users with advanced PC knowledge [10]. After a brief introduction to VIPspace's main features, users had to (i) directly up-/download workspace objects to/from the global workspace via drag & drop (ii) to program a task field to execute an automatic upload and (iii) to move and copy objects between different task fields.

Our results indicate that the separation between private and global workspace by a horizontal bar was clear for each user. Documents could be published from private to public area with no problems. Moving objects between task fields was easy to execute via drag & drop while when copying objects users asked for assistance. This was mainly since it additionally requires the use of context-sensitive menus. Nearly all test subjects classified the specification of rules for programming task fields as intuitive. Especially the use of visual means together with a consistent interaction via drag & drop allowed users to specify more complex rules without any assistance. Merely OR- or AND-relations between basic conditions were mixed up.

Other tests to evaluate the acceptance and expressiveness of the chosen attribute and action icons are planned.

6. Implementation

The VIPspace prototype is implemented in Java. The server uses ObjectStore PSE (Persistent Storage Engine) from ObjectDesign for storing, retrieving, and manipulating its objects. The multi-user access is realized by the use of its transaction mechanism. The VIPspace's client uses JavaSoft's Java Foundation Classes (JFC) - a comprehensive set of GUI components and foundation services for Java development. For Internet mailing, VIPspace uses the final version of the JavaMail 1.1 implementation. The communication between server and client is realized with sockets. Figure 13 gives an overview of the basic architecture.

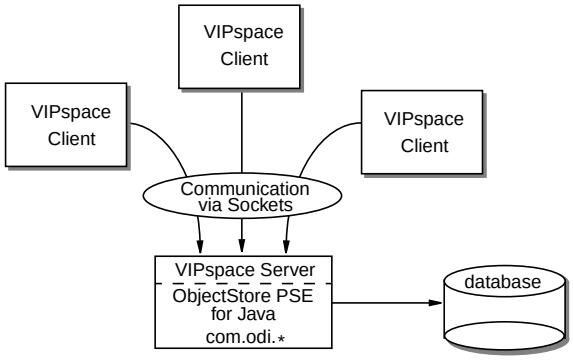

Figure 13. VIPspace Architecture.

7. Conclusions and Future Work

We have introduced the shared workspace VIPspace. VIPspace has been developed as an easy-to-use and easy-to-configure application based on a homogeneous drag & drop paradigm. The various tasks can be configured by the means of a rule-oriented icon-/form-based visual programming language. The paper reflects the current state of the system implemented in Java which provides a portable environment for PC and workstation.

First studies indicate promising results regarding the usability of VIPspace. However, we are still working on improving VIPspace for users with no or only little programming knowledge. We still see some drawbacks in the use of icons which should be intuitive for a wide class of users. There is certainly a trade-off between the number of used icons and intuitive management of them. However, we are quite optimistic that our basic prototype is flexible enough to be easily adapted to the evaluation results of our usability studies.

The main goal of the present VIPspace prototype was the investigation of advanced configuration abilities and user friendly document exchange and processing facilities rather than to provide a complete implementation of a shared workspace. It may be worth to evaluate at this point if VIPspace can be used as a visual interface for already existing workspaces. Thus, our future technical improvements will less concern technical enhancements in the direction of implementing additional underlying services rather than integrating existing ones like videoconferencing, electronic whiteboards, etc.

Acknowledgments

We would like to thank Bernd Gutkauf for various discussions and his valuable remarks when reading this article.

References

[1] D. Coleman and R. Khanna. *Groupware: Technologies and Applications*. Prentice Hall, 1995.

[2] Open Text Corporation. Livelink Intranet Overview. WWW, 1997. (http://demo.opentext.com).

[3] C. A. Ellis, S. J. Gibbs, and G. L. Rein. Groupware: Some Issues and Experiences. *Communications of the ACM*, 34(1), January 1991.

[4] R. Bentley et al. Basic Support for Cooperative Work on the World Wide Web. *International Journal of Human-Computer Studies*, 46(6):827–846, June 1997. Special Issue on Innovative Applications of the World Wide Web.

[5] F. Fluckiger. *Understanding Networked Multimedia Applications and Technology*. Prentice Hall, 1995.

[6] A. Grasso, J.-L. Meunier, D. Pagani, and R. Pareschi. Distributed Coordination and Workflow on the World Wide Web. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 6(2/3):175–200, 1997. Kluwer Academic Publishers.

[7] R. Johansen. *Groupware: Computer Support for Business Teams*. The Free Press, New York, 1988.

[8] D. LaLiberte and D.Woolley. Presentation Features of Text-based Conferencing Systems on the WWW. *Computer-Mediated Communication Magazin*, 5(9), September 1997. (http://www.rpi.edu/ decemj/cmc/mag/ archive.html).

[9] A. Repenning. Bending the Rules: Steps towards semantically enriched graphical rewrite rules. In *Proceedings 1995 IEEE Symposium of Visual Languages*, Los Alamitos, CA, 1995. IEEE Press.

[10] J. Rubart. VIPspace - Ein plattformuebergreifender gemeinsamer Informationsraum mit visuell programmierbarer Benutzungsschnittstelle. Diploma thesis (in german), Paderborn University, 1998.

[11] N. C. Shu. *Visual Programming*. Van Nostrand Reinhold, New York, 1988.

[12] Van Someren, W. Maarten, Y.F. Barnard, and J.A.C. Sandberg. *The Think Aloud Method: A practical guide to modelling cognitive processes*. Academic Press, New York, 1994.

[13] R. Walters. *Computer-Mediated Communications*. Artech House, 1995.

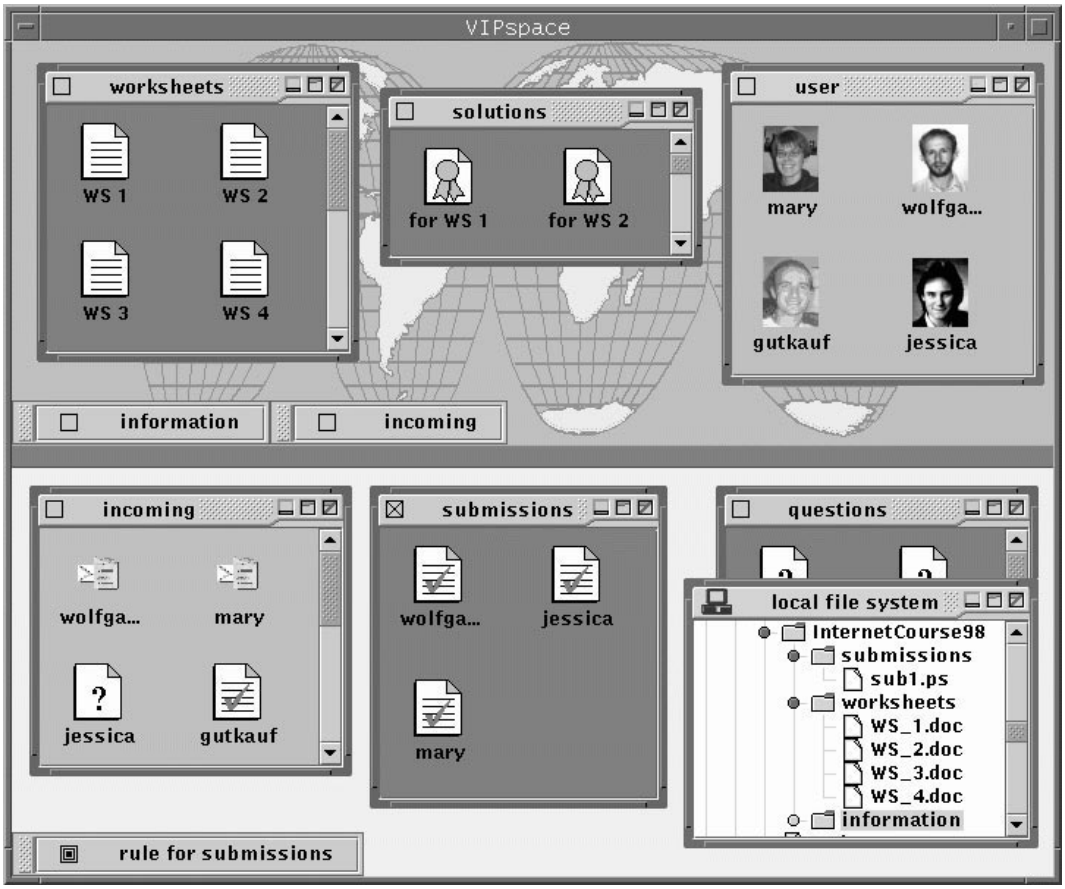

Figure 14. VIPspace Screenshot.