FernUniversität in Hagen

Department of Mathematics
and Computer Science

Extending a shared workspace environment with context-based adaptations

Dirk Veiel, Jörg M. Haake, Stephan Lukosch 2009

Research Report 3/2009
ISSN 1865-3944

© 2009 Dirk Veiel, Jörg M. Haake, Stephan Lukosch

Editor:	Dean of the Department of Mathematics and Computer Science
Type and Print:	FernUniversität in Hagen
Distribution:	http://deposit.fernuni-hagen.de/view/departments/miresearchreports.html

Extending a shared workspace environment with context-based adaptations

Dirk Veiel[1], Jörg M. Haake[1], Stephan Lukosch[2]

[1]Department of Mathematics and Computer Science, FernUniversität in Hagen,
58084 Hagen, Germany
{dirk.veiel, joerg.haake}@fernuni-hagen.de

[2]Faculty of Technology, Policy, and Management, Delft University of Technology,
PO box 5015, 2600 GA Delft, The Netherlands
s.g.lukosch@tudelft.nl

Abstract. Nowadays, many teams collaborate via shared workspace environment which offer a suite of services supporting the group interaction. The needs for an effective group interaction vary over time and are dependent of the current problem and group goal. An ideal shared workspace environment has to take this into account and offer means for tailoring the offered the services to meet the current needs of the collaborating team. Current approaches barely offer means for manual tailoring which is difficult. Context-based adaptation mechanisms can be used to support teams with shared workspace environments best meeting their needs. In this article, we propose a service-oriented architecture of shared workspaces, analyze this architecture to identify adaptation possibilities, introduce the *Context and Adaptation Framework* (*CAF*) as a means to extend shared workspace environment for context-based adaptations and validate our approach by reporting on our prototype implementation.

Keywords: Shared workspaces, adaptation, group context

1 Introduction

In today's global economy, many teams use shared workspace environments providing problem-specific and team-specific tools and artifacts. However, the types of artifacts and tools currently needed may vary over time, task, and phase of collaboration. Thus, shared workspace environments must deal with changing tool sets consisting of single user and cooperative tools, and provide an appropriate integration approach. Team work often leads to changing requirements on the collaboration environment, which must be reflected in its configuration (e.g., available artifacts, sessions, tools).

In order to accommodate such changing needs, the shared workspace environment needs to be adapted. However, such adaptations may be difficult (What is a meaningful adaptation in the current situation? How can I perform such adaptation in the current configuration of the shared workspace?) and cause overhead. Thus, the collaboration environment should make adaptations as easy as possible. Furthermore, changing collaboration situations may require adaptations of multiple tools

and artifacts. E.g., if authors begin to share a document, the editor should be augmented with awareness and communication functionality to support emerging collaboration.

Current approaches support manual tailoring of shared workspaces (e.g. in CURE [12, 13], BSCW [3]) either in terms of artifacts and workspace structure or they focus on adaptation of single tools to the needs of the individual user or, in some cases, to the needs of a group. However, there is no support for adapting a shared workspace environment to the needs of a team in terms of, firstly, a changing set of collaboratively used tools and artifacts, and, secondly, supporting complex adaptations across several tools and artifacts.

We propose an extended shared workspace environment supporting context-based adaptation, which helps to minimize adaptation overhead, and which supports adaptations affecting multiple tools to better match the users' current needs. At the core of our approach is the *Context and Adaptation Framework (CAF)*, which facilitates the conversion of tools into context-adaptive tools providing the necessary interfaces to our run-time adaptation environment, and a run-time environment executing adaptation rules, which lead to modified workspace configuration and tool behavior.

In the next section, we briefly examine related work. In section 3, we propose a service-oriented architecture of shared workspaces and analyze possible points of adaptation. Section 4 presents our approach of specifying adaptation behavior as rules, introduces our run-time adaptation engine, and presents the *Context and Adaptation Framework (CAF)* as a means to make tools ready for context-based adaptation. Section 5 briefly presents our prototype implementation, which is the basis for the validation results discussed in section 6.

2 Related Work

We start this section by reviewing relevant shared work environments and discuss how these systems deal with possible adaptations. After that, we take a look at context-adaptive systems in general. We review whether the taken approaches are suitable to support context-adaptive collaboration in shared work environments.

BSCW [3] and *CURE* [12, 13] are web-based shared work environments that offer a variety of collaboration services, e.g. communication and document sharing. *CHIPS* [24] is a cooperative hypermedia system with integrated process support. *TeamSpace* [10] offers support for virtual meetings and integrates synchronous and asynchronous team interaction into a task-oriented collaboration space. *BRIDGE* [8] is a collaboratory that focuses on supporting creative processes and as such integrates a variety of collaboration services. All of the above examples focus on a specific application domain and only offer a fixed set of services to the user. Some of them, e.g. *CHIPS* or *CURE* are highly tailorable, but they do not automatically adapt their offered functionality to improve the collaboration within a team.

The most prominent examples for context-based adaptation focus on single users and consider location as most relevant context information (e.g., [1, 14, 21]) or focus on learner profiles (cf. ITS). Compared to single-user ITS, *COLER* [6] provides a software coach for improving collaboration. *CoBrA*

[4, 5] is an agent-based architecture that uses shared context knowledge represented as a ontology to adapt service agents according to a user's context. Gross and Prinz [11] introduce a context model and a collaborative system that supports context-adaptive awareness. The context model consists of events, artifacts, locations, etc. The main restrictions of their approach are that the context representation is only used to update and visualize awareness information and that only one cooperative application can be used. Edwards [7] explores the space between two different context understandings: in CSCW research, people are assumed to be the consumer of context information; the ubiquitous computing community has the opinion that systems are the consumers of context information. *Intermezzo* [7] tries to fill this gap through the creation of new high-level services. However, Intermezzo does not offer an approach to integrate and use these services within a shared workspace. Rittenbruch describes an approach to the representation of context of awareness information but real world examples are missing [20]. Fuchs [9] describes an integrated synchronous and asynchronous notification service for awareness information called *AREA*, but again *AREA* uses the context representation only for awareness information. Ahn et al. [2] introduce a knowledge context model. Based on this context model they implement the *virtual workgroup support system (VWSS)*. However, VWSS does not focus on improving the interaction of the users by adapting the workspace functionalitiy. One drawback of their solution is that their knowledge context model has to be extended for other application domains. The *Semantic Workspace Organizer (SWO)* [19] is an extension of *BSCW*. It analyzes user activities and textual documents inside the shared workspace to suggest appropriate locations for new document upload and for document search. The *ECOSPACE* project aims at providing an integrated collaborative work environment [17, 18]. For that purpose, *ECOSPACE* uses a service-oriented architecture and provides a series of collaboration services for orchestration and choreography. The orchestration and choreography is based on an ontology which still has to be described [17, 23].

The above approaches focus on adaptations which are used in specific domains, e.g., single-user systems or ITS, or on sub-domains in the field of CSCW, e.g. awareness or knowledge management. Adaptation based on group context and for multiple users of a cooperative system is intended only by *ECOSPACE*, but the required context model is still an open issue. Similarly, only *ECOSPACE* supports the integration of different collaboration services within the same shared work environment. In summary, current approaches do not provide a sufficient context model for adapting the collaboration in shared workspaces and do not use the context information to adapt the interaction of the users.

3 Possible adaptations in service-oriented shared workspaces

A collaborative application typically uses the model-view-controller paradigm [15] and can be based on a client-server architecture. Nowadays, service-oriented client-server architectures are used to create collaborative services to be consumed by collaborative applications (i.e. clients). In Figure 1 we address these kinds of architectures and split them up into four layers: *UI Layer*, *Logic Layer*, *Services Layer*, and the *Model Layer*. The view and controller parts of the tools (Application$_1$ UI, Application$_2$ UI) are usually running on the client side (addressed by the *UI Layer*). The *Logic Layer* consists of the application specific business logic (*Application$_1$ Logic, Application$_2$ Logic*) that can be built by using different services from the underlying *Services Layer*. The *Services Layer* contains

Application Services as well as *Collaboration Services* and is used by the above layer to accesses the artifacts ($Artifact_1$, $Artifact_2$ and $Artifact_3$) represented in the *Model Layer*. If more than one application (consisting of *Application UI* and *Application Logic*) is present, we talk about a shared workspace environment.

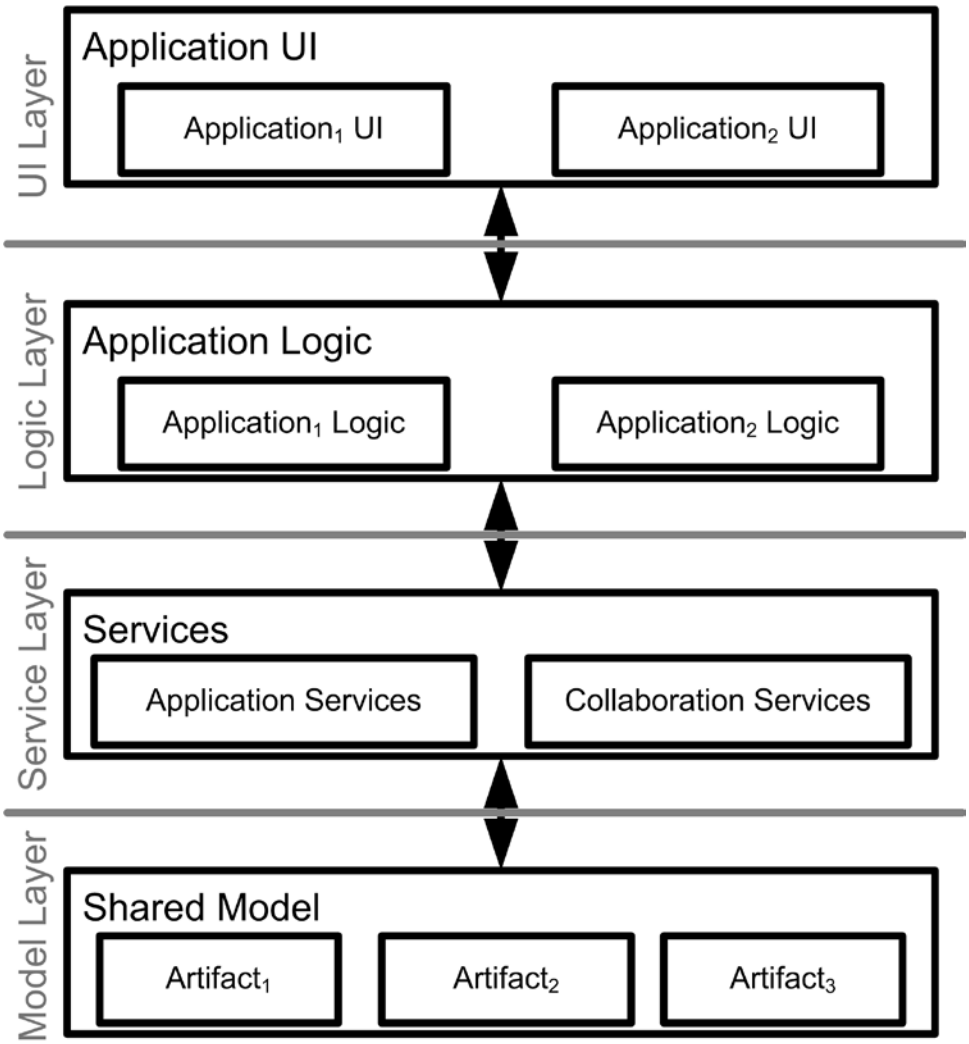

Figure 1 Service-oriented architecture of a shared workspace environment

We use this architecture of service-oriented shared workspaces to examine possible points for adaptation, which may be used to adapt the behavior of such workspaces at run-time. Following the layers one can distinguish four areas of adaptation in this architecture:

1. *Modification of the application user interface:* Usually, the UI can be modeled as a hierarchy of visual components (views and controllers). Thus, the following manipulations can be used for adaptation purposes: (i) add, remove, or replace entire visual components in the hierarchy (e.g., replace a simple user list widget by a more sophisticated radar view), and (ii) replace individual views or controllers (e.g., change the supported interaction style by replacing the controller of a visual component).

2. *Modification of application logic:* Here, we may change the internal structure of the application logic by adding, removing, or replacing application services used by the application, or by changing the execution structure of the application services. In addition, entire new applications (i.e. application logic) could be added or removed in this layer.

3. *Modification of services:* In this layer, application services as well as collaboration services can be added or removed (e.g., by adding an audio-based communication channel or removing instant messaging functionality from the current workspace). This impacts the way users may communicate, coordinate and share objects. In addition, running services could be replaced (e.g., replacing an audio-based communication channel by a text-based one).

4. *Modification of shared model:* Finally, the shared model layer could be adapted by manipulating artifacts and sessions. Artifacts respectively documents could be created or deleted and their attributes could be manipulated. Sessions could be created or closed, and members, tools, or artifacts could be added to or removed from a session. In addition, attributes of sessions may be manipulated.

In the following, we use the adaptation possibilities identified above to firstly define an architecture for specifying and implementing such adaptations of shared workspaces and secondly a framework supporting developers in extending applications with adaptation functionality (i.e. providing the necessary interfaces for interaction with our proposed adaptation architecture).

4 Adaptation in service-oriented collaborative applications

4.1 Specification of adaptation behavior

In the following, we briefly introduce a context model for defining conditions for adaptation in our *Context and Adaptation Framework* (*CAF*) [16]. This context model addresses scenarios in which several actors collaborate to achieve a shared goal and thereby captures basic concepts of co-located and distributed collaboration. Still, the context model is completely open for extensions that cover further collaboration aspects.

Figure 2 summarizes our context model and shows the basic context classes[1] and their relations. An *Application* implements the model-view-controller (MVC) paradigm [15] and consists of *Views* and *Controllers* components. *Views* and *Controllers* use *Services* to access the *Artifacts*. *Artifacts* use *Services* to notify *Views* and *Controllers* about changes. Each *Application* is part of a *User Workspace* and is created by an *Application Factory* which specifies what *Applications* are available within a workspace and how these can be initialized. Each *Actor* has a *User Workspace* and belongs to at least one *Team*. The *User Workspace* defines the *Roles* of an *Actor* within this *Team*. Each *Role* allows an *Actor* to perform specific *Actions* within an *Application*. The available *Actions* within an *Application* are defined by its *Application Functionality*. *Actors* then interact with the *Application* by performing *Actions* allowed by their *Roles*. These *Actions* are received by the corresponding *Controller* components of the *Application*.

[1] Please note, in the following context classes are set in *italics*.

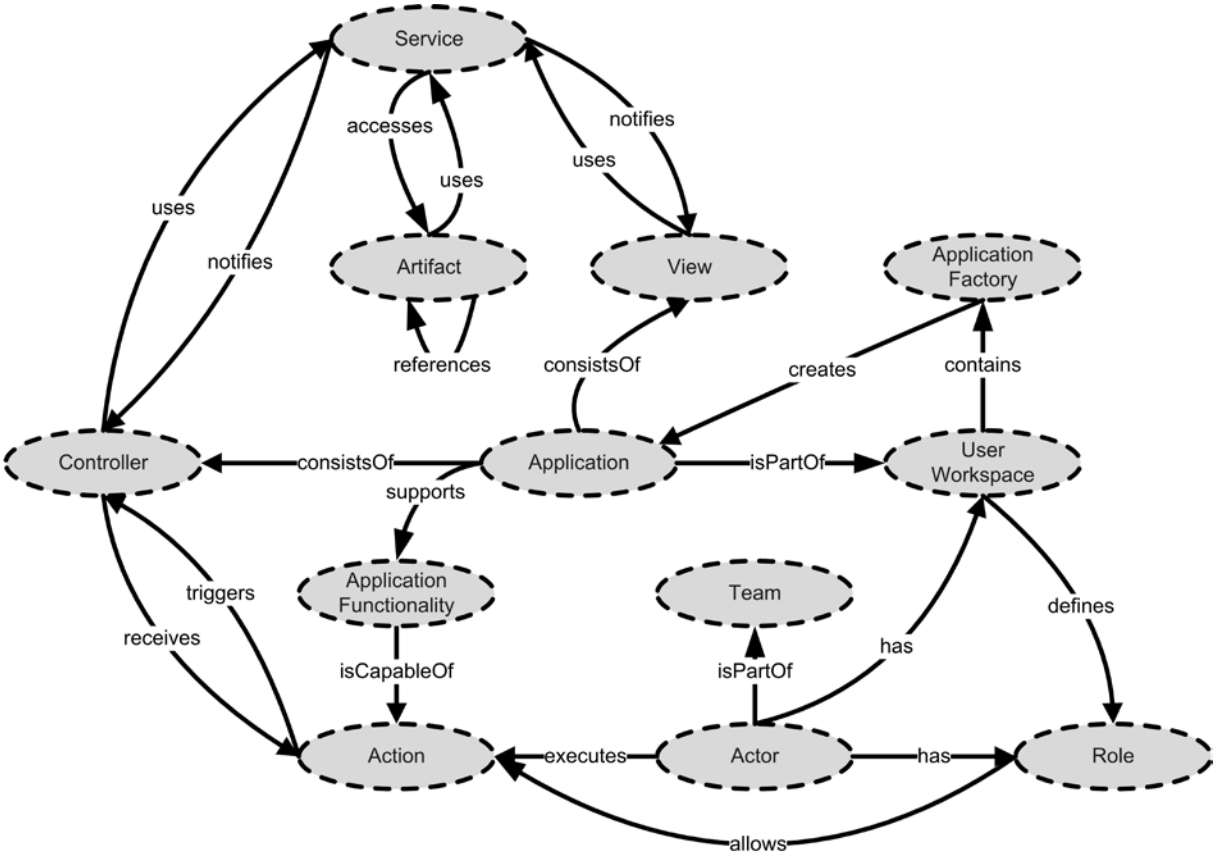

Figure 2 Collaboration context model

The *Application Functionality* class has several subclasses not shown in Figure 2, e.g., *Communication*, *Shared Editing*, *Awareness*, *Management* or *Workflow Management*. All of these classes are specialized into further subclasses. The *Communication* concept, e.g., distinguishes between *Synchronous* and *Asynchronous Communication*. The class *Synchronous Communication* then distinguishes between *Audio*, *Video*, or *Chat*. Similarly, the *Awareness* class distinguishes between *Synchronous* and *Asynchronous Awareness*. The class *Synchronous Awareness* then distinguishes between e.g., *Active Neighbors*, *Activity Indicator*, *Remote Field of Vision*, *Remote Cursor*, *Telepointer*, or *User List*. The *Management* class, e.g., distinguishes functionality for *Access Right*, *Session*, *User,* or *Concurrency Control Management*. The *Shared Editing* class distinguishes different kinds of editors for, e.g., *Text*, *Rich Text*, *Image* or *Calendar*. Most of these classes are derived from patterns for computer-mediated interaction [22] which describe best practices for designing tools for collaboration. Figure 3 shows an excerpt of the above class hierarchy and highlights the *Chat* application functionality. Thus, each application which supports chat application functionality has to offer at least two action types: *OpenChat* and *SendMsg*.

All above classes are necessary to model the configuration of shared workspaces and tools and to capture the current context at runtime. As a sample scenario consider that a team consisting of Alice and Bob synchronously collaborates on a shared text document. We assume, that Alice and Bob created user workspaces sharing the design artifacts (i.e. documents). Alice then created a shared text document and opened a shared text editor to work on a design document. While Alice has the role of an author, Bob has the role of a reviewer. Bob later opened the same shared text document

to review the current state of the design. Both team members have different roles highlighting their tasks within the team.

Figure 3 Application functionality concepts and relations

Figure 4 shows the current context state for the above scenario. For space reasons we omit view and controller instances (which can be reached via the respective application instances) as well as the relations between the different actions and the corresponding application functionality. Different adaptation possibilities exist to improve the interaction between Bob and Alice, e.g., to provide additional awareness information, to enable concurrency control mechanisms, or to establish a communication possibility. Choosing a good adaptation in such a situation is difficult and is highly dependent on the context and interaction history of the team.

Given the context state described in Figure 4, establishing a communication possibility among the two actors seems a good adaptation possibility. The workspaces of both actors offer an application factory for a chat tool. Thus, we propose that a corresponding adaptation rule checks the available applications within each user's workspace and whether the current roles allow the actors to communicate with each other. The following pseudo code shows an adaptation rule which makes use of the current context state and adapts the users' workspaces:

```
rule "open communication channel"
   when
      artifacts: getArtifactsInContext("OpenText:bob")
      actors: getActorsInContext(artifacts)
      communication: getApplicationsInContext(actors,
         "Synchronous Communication")
```

```
      selectedApplication: selectOneFrom(communication)
   then
      openForAll(selectedApplication, actors)
end
```

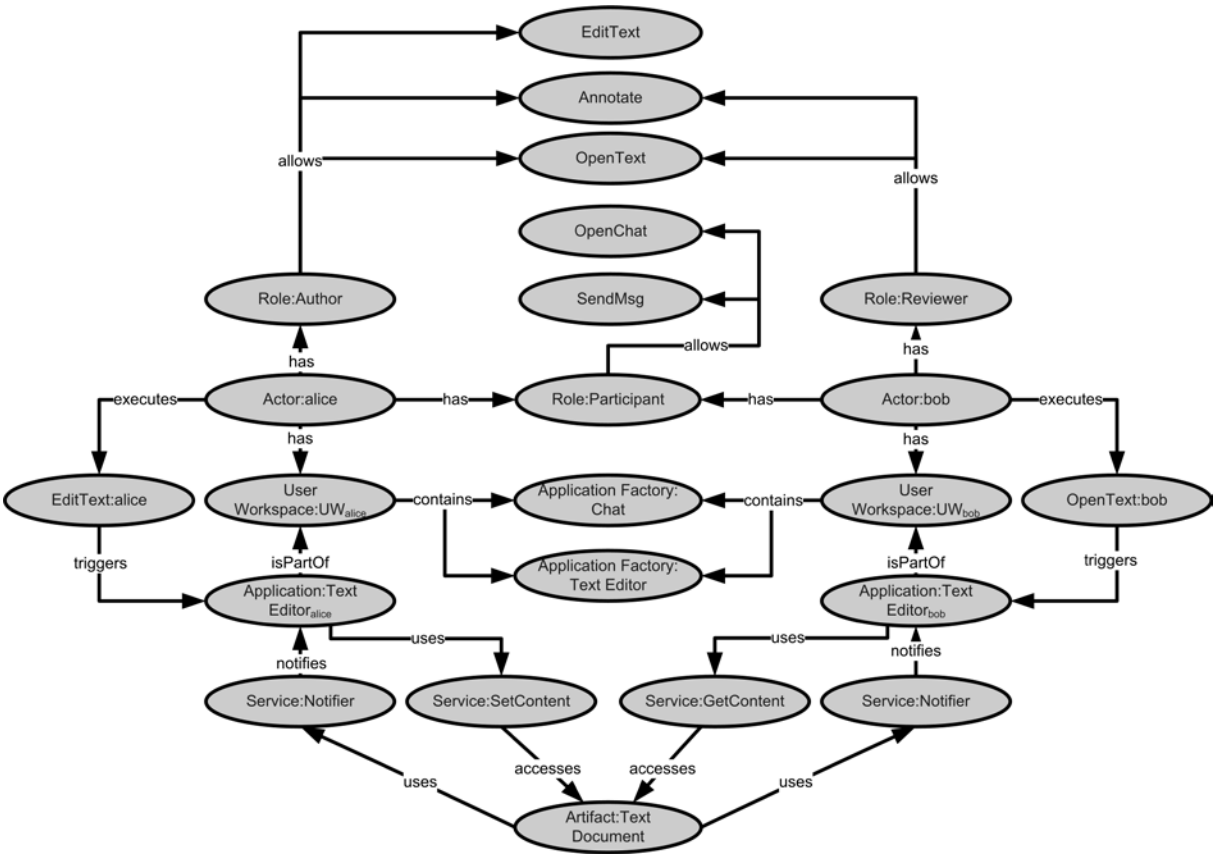

Figure 4 Sample context state before adaptation

The above adaptation rule consists of a condition part and an action block. The action block may contain several adaptation actions in order to facilitate cross-application adaptations. By, e.g., adding a service to the current application configuration the adaptation action may change the current context state. In the above example, the condition is triggered by *Actor:bob* opening the *Artifact:Text Document*. The action part of the above adaptation rule is executed if all conditions are valid, i.e. in our example none of the retrieved result sets are empty. In this adaptation rule, `getArtifactsInContext` returns a set of artifacts which are in the context of the action *OpenText:bob*. The function `getActorsInContext` then calculates all actors which access the artifacts in the context of *OpenText:bob*, i.e. in Figure 4 *Actor:alice*. The function `getApplicationsInContext` then determines in this case all applications which support the application functionality *Communication* and are connected to all actors accessing the same artifacts, i.e. in Figure 4 *Application Factory:Chat*. The function `selectOneFrom` selects from a set of context elements the one which has been used most by the collaborating actors. The corresponding information is stored as preference value with the different edges of the context graph and is updated via a special learning algorithm. Figure 5 shows the context state after applying the adaption rule.

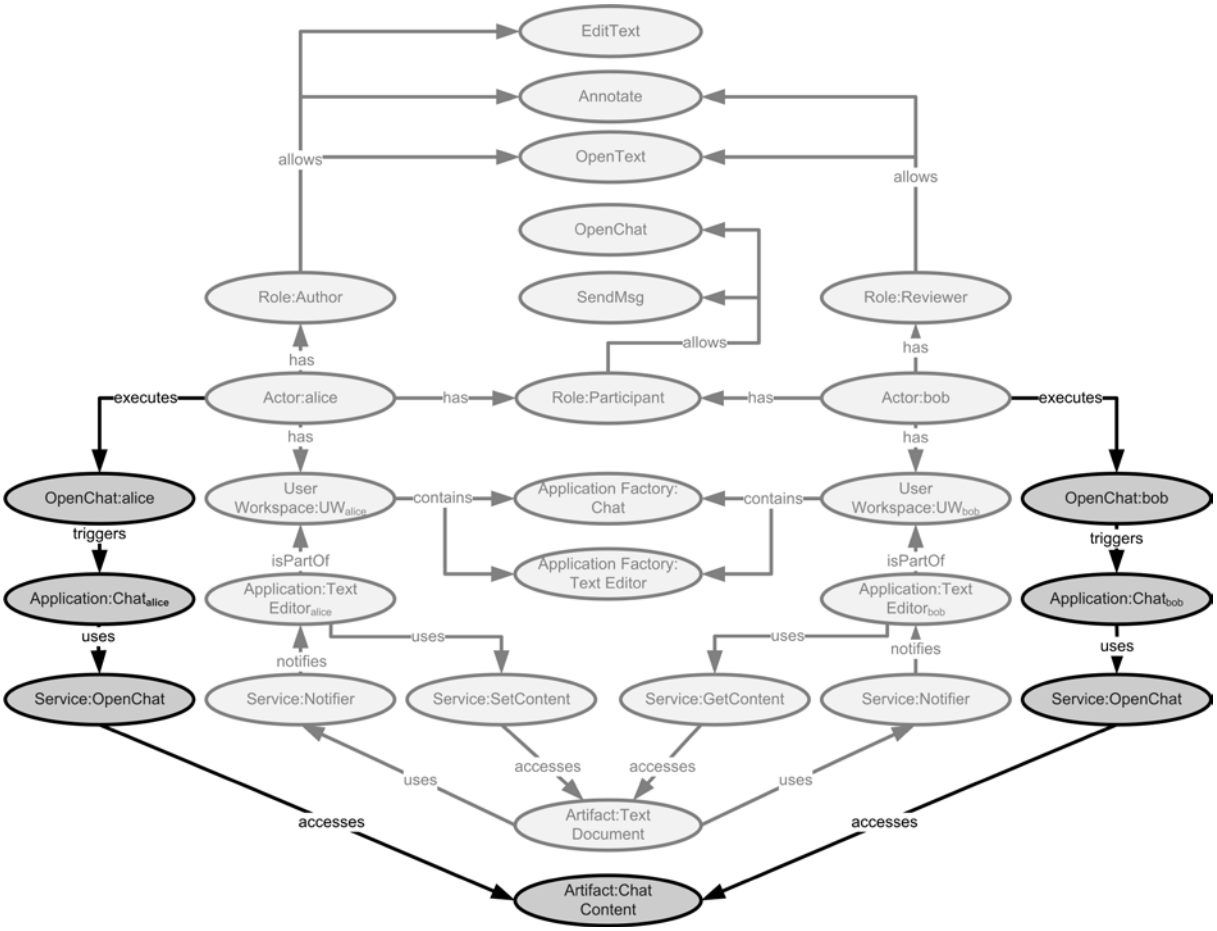

Figure 5 Sample context state after adaptation (previously existing instances and relations are shown in light grey)

The above adaptation rule is only one example for a possible adaptation for the given context. There exist further adaptation possibilities to improve the interaction within a team, e.g., to provide additional awareness information or to enable concurrency control mechanisms. Choosing a good adaptation in such a situation is difficult and is highly dependent on the context and interaction history of the team. The following adaptation rule shows how awareness could be improved by enabling an synchronous awareness widget when at least one additional actor accesses an artifact in the context of *OpenText:bob*:

```
rule “open synchronous awareness widget”
   when
      artifacts: getArtifactsInContext(“OpenText:bob”)
      actors: getActorsInContext(artifacts)
      applications: getApplicationsInContext(actors, artifacts)
      awareness: getFunctionalityInContext(application,
         “Synchronous Awareness”)
      selectedApplication: selectOneFrom(awareness)
   then
      openForAll(selectedApplication, actors)
end
```

Finally, let us consider the adaptation which would enable a concurrency control mechanism. In this case, the *Application:Text Editor* would have to support the corresponding application functionalities concerning concurrency control management, e.g. *Optimistic Concurrency Control*, *Pessimistic Concurrency Control*, or *Operational Transformation*. An adaptation rule, triggered by *OpenText:bob*, would again first check, whether other actors access the artifacts in the context of *OpenText:bob* and in the positive case look for available concurrency control mechanisms. From the available mechanisms, one would be chosen according to the preferences of the collaborating actors.

As the above examples show, we use the current context information to recognize specific situations, which show potential for improvements as expressed in the rule's condition, and perform the adaptation described in the action part of the rule. In the above example we consider the situation where at least two users share the same artifact at the same time and assume that a communication channel would help to coordinate the users' work. Automated rule execution minimizes the users' adaptation efforts. Obviously, more generic rules (i.e. independent from specific users) are applicable in more cases and thus express general policies, while more specific rules can be used to express user or team specific preferences.

4.2 Making tools ready for adaptation

Next we address the question of how to make either existing or new tools in a shared workspace environment ready for adaptation. From a developer's point of view, integrating an application into our *Context and Adaptation Framework (CAF)* should be as simple as possible as well as minimize the implementation effort. These are the two requirements we have considered while designing the framework.

We propose an approach that extends usual client-server applications following the architecture as described in Section 3 by adding components of our *CAF* shown in Figure 6 in grey. Thus, our approach also distinguishes the four layers: *UI Layer*, *Logic Layer*, *Service Layer* and *Model Layer*.

The *UI Layer* contains the *Application UI*s and the *Adaptation Component* of *CAF*. We use the *Adaptation Component* to start and stop *Application UI*s, or to use a specific interface an application offers to apply adaptation actions to the *Application UI*. Currently, this interface contains methods to show or hide a certain GUI component, to set the focus to a specific GUI component, to modify the content of a GUI component (e.g. text of a label, button), to highlight a specific GUI component (e.g. by enlarging the font, changing the sort order or filtering option of a list, marking a text, playing a sound, changing the color), to maximize or minimize the view, to set the read-only mode, to scroll to a certain position, or to lock the scrollbars (e.g. in case of a tightly coupled shared editing session). We are going to extend this interface while proceeding with our work.

Within the *Logic Layer* you will find the *Application Logic* as well as the *Adaptation Server*. Our *CAF* adds the components *Basic Services* and *Notifier* to the *Application Logic*. The *Application Logic* can use *Basic Services* to integrate the application into the *CAF* or use the *Notifier* to send notifications to the client side. The developer can use *Basic Services* to integrate support for, e.g., sensing functionality, access right and user management, multiple service provider support, login and logout functionality, access to a database service, or action-based configuration management into existing services and thus, build a bridge to the *Adaptation Server*. Furthermore, the *Basic Services* can call

the *Application Logic* of specific applications to call services (e.g. to change the configuration of the service). The *Adaptation Server* is based on a service-oriented architecture and hosts components like *Sensing*, and the *Adaptation Engine*. The *Sensing* component uses the information about service calls and user interactions (given by the *Basic Services*) to update the current context representation. The *Adaptation Engine* uses the current context representation to find corresponding adaption rules and to execute them. Adaptation actions can affect *Application UI*, *Application Logic*, *Services*, and the *Shared Model*.

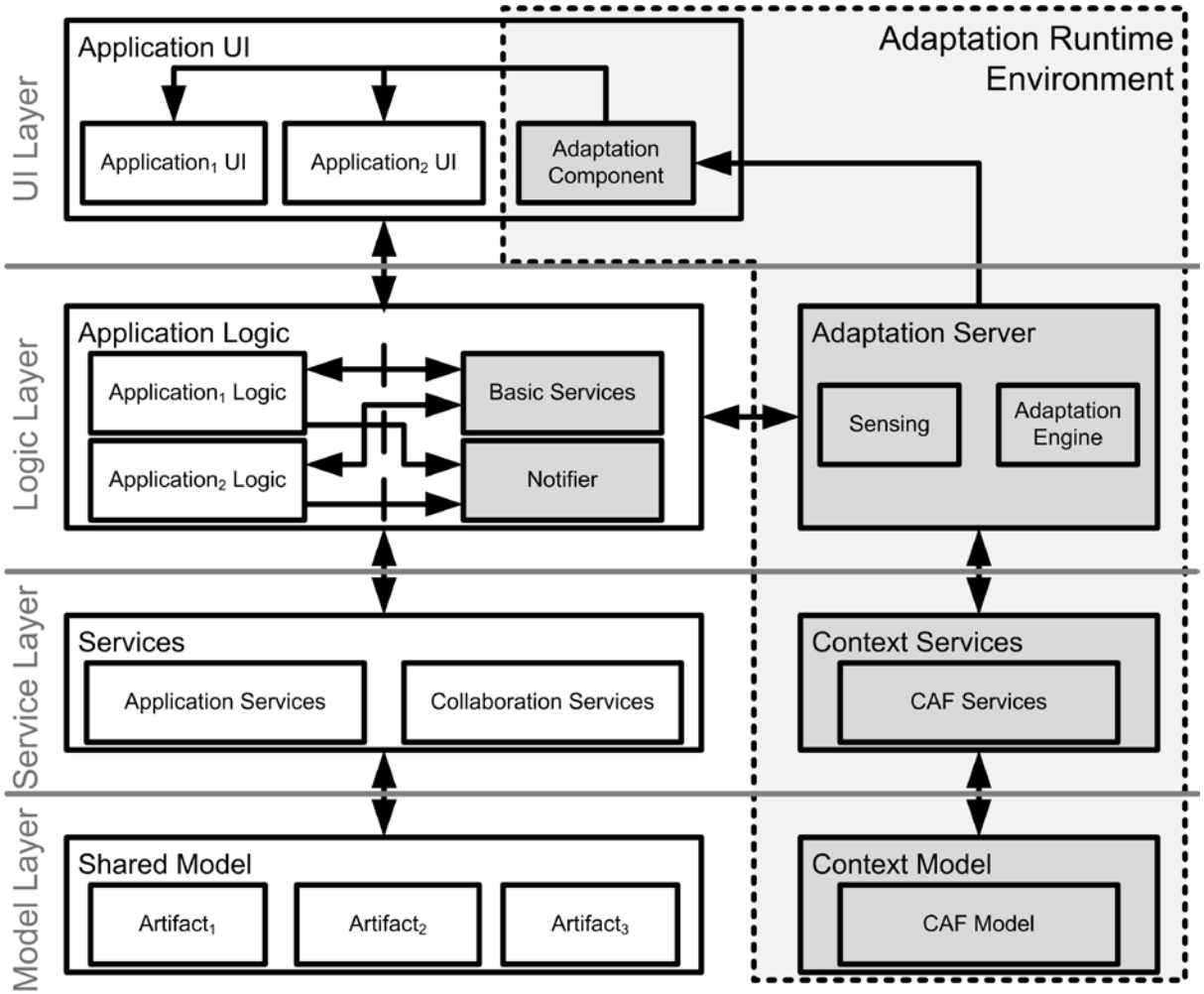

Figure 6 Extended usual client-server application by Context *and Adaptation Framework (CAF)*

The next layer is the *Service Layer*. It is used by the *Logic Layer* to access the *Model Layer* and contains *Services* (on the left hand side) and *Context Services* (on the right hand side). The *Services* include *Application* and *Collaboration Services* (cf. Section 2).

The bottom layer is the *Model Layer*. It contains the *Shared Model* (on the left hand side) and the *Context Model* (on the right hand side). The *Context Model* contains the current context representation and the adaptation knowledge (e.g., adaptation rules, list of currently applied adaptation rules).

Our *Adaptation Runtime Environment*, shown on the right hand side of Figure 5, allows the execution of adaptation rules, which adapt the configuration of shared workspace environments and the behavior of applications offering our adaptation interface. It can be split up into the layers *Adaptation Server*, *Context Services* and *Context Model*.

Runtime adaptations of an application require that the developer implements an adaptation and a service interface (usually this interface exists) and registers the service interface of the application at *CAF* by using *Basic Services*. The adaptation interface (*Application UIs*) is used by the *Adaptation Component* for adaptation of the UI parts. The implementation of this interface has to map the corresponding method calls to changes at the UI (e.g. show or hide a given GUI component). The service interface usually exists because it is the service interface of the application logic. This interface has to be registered at *CAF* by using *Basic Services*. This is necessary to get sensing information from the service calls of an application (within the *Application Logic* layer). This information is used to update the context representation that is used to find possible adaptation rules and execute them. Furthermore, the last mentioned interface is used in case of service configuration adaptations (e.g. changing the concurrency control algorithm).

Our approach supports, firstly, context-based adaptation across multiple tools, since all tools provide the required standard interfaces, and secondly, a rule syntax to express multiple adaptation actions on multiple tools and artifacts.

4.3 Executing the rules at run-time

A flexible adaptive system executes a cycle of

1. User interaction
2. Sensing user activities
3. Adapting system behavior
4. Modifying adaptation knowledge (e.g., if users want to change adaptation rules)

The user interacts with the tools of the shared workspace environment. Usually, these interactions imply service calls that can be sensed (e.g. by using our *Basic Services*) to update the context representation of the current situation. The current context representation is used by the *Adaptation Engine* to find corresponding adaption rules and execute them. An adaptation rule can modify or change the configuration of components of the following layers: *UI Layer*, *Logic Layer*, *Service Layer* and *Model Layer*. Furthermore, the current context representation itself can be modified by applying adaption rules.

Typical adaptations of which a user is aware effect the UI and, e.g., show or hide a certain GUI component, set the focus to a specific GUI component, modify the content of a GUI component, highlight a specific GUI component, minimize or maximize the view, set the read-only mode, scroll to a certain position, or lock the scrollbars. Adaptations that affect the *Logic Layer* may change the service composition (e.g. the used concurrency control algorithm). Starting or stopping a service (e.g. the chat service in the above example) can be a possible adaptation at the *Service Layer*. At the

Model Layer the adaptations can reach from changing attributes of an artifact to creating or deleting artifacts to, .e.g., provide scaffolding structures for improving group interaction. All of the aforementioned adaptations are supported by *CAF* and can be part of the action part of an adaptation rule.

4.4 Implementation

We implemented the conceptual architecture shown in Figure 6 to support context-based adaptations within a shared work environment. In the current prototype, the *Application Logic* and the *Adaptation Server* are based on Equinox[2] and realize all components as so-called bundles in *OSGi*[3]. In the prototype of the *Application Logic*, we integrated two *Applications*. Firstly, we adapted *CURE* [12, 13] to provide *Application Functionality* for *Document*, *User*, and *Workspace Management* as well as *Asynchronous Awareness* and *Communication*. Secondly, we used *R-OSGi*[4] to develop and integrate Application Functionality for *Synchronous Awareness* and *Communication*. The *UI*-parts for these service classes are implemented as plug-ins for *Eclipse*[5]. We use Drools[6] as an adaptation engine and the corresponding rule syntax to define adaptation rules.

As mentioned in Section 4, a developer has to register the service interface of an application by using the *Basic Services*, e.g., to integrate sensing functionality into the application. We use the Java Reflection API for dynamic proxy creation of the registered service interface to get the service calls, and Java annotations to get the corresponding information that the context should reflect. We are using Java annotations because they are easy to integrate into existing interface definitions and the integration effort is low. An example of an annotated method that needs read rights on the specified workspace looks like this:

```
@Operation(type=READ)
void openWorkspace(
  @SessionKey() String sessionKey,
  @Artifact(type=WORKSPACE, src=URI) String wsURI);
```

The proxy performs the following steps: 1) Receive the service calls from the client side. 2) Interpret specific Java annotations (e.g. `@Operation`, `@SessionKey`, `@Artifact`) that *CAF* supports of the corresponding method at runtime and use it to retrieve the necessary arguments from the list of arguments (needed by the following steps). 3) Check the validity of the session key (the session key is specified by the annotation `@SessionKey`). 4) Check the access rights (arguments are specified by the annotations `@Operation`, `@SessionKey` and `@Artifact`). 5) Send the context information to the *Sensing* component of the *Adaptation Server* (i.e. the user specified by the given session key opens the workspace specified by the given workspace URI). 6) Call the implementation of the method and return the results. We minimize the integration effort for developers (i.e. lines of code and learning curve) by moving the steps 2) to 5) into our proxy implementation, i.e. the developer does not has to write the code for these steps.

[2] http://www.eclipse.org/equinox/
[3] http://www.osgi.org
[4] http://r-osgi.sourceforge.net/
[5] http://www.eclipse.org/
[6] http://www.jboss.org/drools

5 Validation

In order to support automatic adaptation of shared workspaces to the changing needs of collaborating users we developed (1) the conceptual architecture of adaptive shared workspace systems, (2) a context model for shared workspace systems and a matching adaptation rule syntax, (3) the *CAF* framework supporting the conversion of service-oriented applications into adaptive collaborative applications, and (4) the adaptation runtime environment for executing adaptation rules.

We validated our context model and rule syntax by modeling typical collaboration situations and by modeling adaptation rules that seem useful in these situations (cf. Section 4.1).

We prototypically implemented *CAF* and our conceptual architecture (cf. section 5) and used it to integrate a number of collaborative service-oriented tools. Examples include *CURE* [12, 13] which was integrated to provide *Application Functionality* for *Document*, *User*, and *Workspace Management* as well as *Asynchronous Awareness* and *Communication*, and *R-OSGi*[7] which was used to develop and integrate Application Functionality for *Synchronous Awareness* and *Communication*. The *UI*-parts for these service classes were implemented as plug-ins for *Eclipse*[8]. These experiences show that our approach can be successfully used to convert standard tools into context-adaptive tools. Experiences from the conversion process show that the approach is simple and developers can apply it with small effort.

Functional tests demonstrated that adaptation rules for typical collaboration situations can in fact be implemented and executed in our prototype, leading to meaningful adaptations at the UI, application logic and shared model layers.

By testing adaptation rules affecting two applications (such as changing the displayed awareness information in an editor or establishing communication channels between users) we tested that our approach supports cross-application adaptation.

Together, these experiences show that our approach provides a context model sufficient for adapting collaboration among users of a shared workspace and can exploit this context information to adapt the interaction among users and between users and their tools.

6 Conclusions

In this paper we proposed a service-oriented architecture of shared workspaces and analyzed possible points of adaptation on four layers (*UI Layer*, *Logic Layer*, *Service Layer*, *Model Layer*). We introduced a context model and an adaptation rule syntax for expressing context-based adaptations of shared workspaces. Using *CAF*, we support the integration of service-oriented applications into our adaptation runtime environment. We introduced our prototypical implementation and presented our validation results.

[7] http://r-osgi.sourceforge.net/
[8] http://www.eclipse.org/

Our approach exceeds the state of the art (cf. Section 2) in several ways: firstly, we provide a context model sufficient for adapting collaboration among users of a shared workspace and, secondly, our approach can exploit this context information to adapt the interaction among users and between users and their tools. Finally, our approach is open for inclusion of new services, applications, and adaptation rules. By extending the context model, new rules can be introduced and address new collaboration aspects and situations without rendering old rules meaningless.

Currently, our prototype implementation is used for functional testing and evaluation in pilot test cases. For the near future, we plan to include applications for adaptation rule tracing, editing and negotiation; testing of adaptation rules in real work situations with the goal of identifying good adaptation practice; and performance tuning of rule execution.

7 References

1. Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: a mobile context-aware tour guide. *Wireless Networks*, 3(5):421–433, 1997.

2. Hyung Jun Ahn, Hong Joo Lee, Kyehyun Cho, and Sung Joo Park. Utilizing knowledge context in virtual collaborative work. *Decision Support Systems*, 39(4):563–582, 2005.

3. W. Appelt and P. Mambrey. Experiences with the BSCW shared workspace system as the backbone of a virtual learning environment for students. In *Proceedings of ED-MEDIA99*, 1999.

4. Harry Chen, Timothy W. Finin, and Anupam Joshi. Using owl in a pervasive computing broker. In Stephen Cranefield, Timothy W. Finin, Valentina A. M. Tamma, and Steven Willmott, editors, *Proceedings of the Workshop on Ontologies in Agent Systems (OAS 2003)*, pages 9–16, 2003.

5. Harry Chen, Timothy W. Finin, and Anupam Joshi. Semantic web in the context broker architecture. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004)*, pages 277–286. IEEE Computer Society, 2004.

6. Mará de los Angeles Constantino-González and Daniel D. Suthers. Automated coaching of collaboration based on workspace analysis: Evaluation and implications for future learning environments. In *Proceedings of the 36th Hawai`i International Conference on the System Sciences (HICSS-36)*. IEEE Press, 2003.

7. W. Keith Edwards. Putting computing in context: An infrastructure to support extensible context-enhanced collaborative applications. *ACM Trans. Comput.-Hum. Interact.*, 12(4):446–474, 2005.

8. Umer Farooq, John M. Carroll, and Craig H. Ganoe. Supporting creativity in distributed scientific communities. In *GROUP '05: Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, pages 217–226, New York, NY, USA, 2005. ACM.

9. Ludwin Fuchs. AREA: a cross-application notification service for groupware. In *ECSCW'99: Proceedings of the sixth conference on European Conference on Computer Supported Cooperative Work*, pages 61–80. Kluwer Academic Publishers, 1999.

10. Werner Geyer, Heather Richter, Ludwin Fuchs, Tom Frauenhofer, Shahrokh Daijavad, and Steven Poltrock. A team collaboration space supporting capture and access of virtual meetings. pages 188–196, Boulder, Colorado, USA, 2001. ACM Press, New York, NY, USA.

11. Tom Gross and Wolfgang Prinz. Modelling shared contexts in cooperative environments: Concept, implementation, and evaluation. *Computer Supported Cooperative Work (CSCW)*, 13(3):283–303, August 2004.

12. J. M. Haake, A. Haake, T. Schümmer, M. Bourimi, and B. Landgraf. End-user controlled group formation and access rights management in a shared workspace system. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 554–563. ACM Press, New York, NY, USA, November 2004.

13. J. M. Haake, T. Schümmer, A. Haake, M. Bourimi, and B. Landgraf. Supporting flexible collaborative distance learning in the CURE platform. In *Proceedings of the Hawaii International Conference On System Sciences (HICSS-37)*. IEEE Press, January 2004.

14. Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra, and Mirjana Spasojevic. People, places, things: web presence for the real world. *Mobile Network Applications*, 7(5):365–376, 2002.

15. Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, August 1988.

16. Stephan Lukosch, Dirk Veiel, and Jörg M. Haake. Enabling context-adaptive collaboration for knowledge-intense processes. In José Cordeiro and Joaquim Filipe, editors, *Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS 2009)*, volume HCI, pages 34–41. INSTICC – Institute for Systems and Technologies of Information, Control and Communication, Portugal, 2009.

17. M. A. Martínez-Carreras, A. Ruiz-Martínez, F. Gómez-Skarmeta, and W. Prinz. Designing a generic collaborative working environment. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 1080–1087, 2007.

18. W. Prinz, H. Loh, M. Pallot, H. Schaffers, A. Skarmeta, and S. Decker. ECOSPACE – towards an integrated collaboration space for eprofessionals. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 39–45, 2006.

19. Wolfgang Prinz and Baber Zaman. Proactive support for the organization of shared workspaces using activity patterns and content analysis. In *GROUP '05: Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, pages 246–255. ACM, New York, NY, USA, 2005.

20. Markus Rittenbruch. Atmosphere: towards context-selective awareness mechanisms. In *HCI (2)*, pages 328–332, 1999.

21. Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *First Annual Workshop on Mobile Computing Systems and Applications (WMCSA)*, December 1994.

22. Till Schümmer and Stephan Lukosch. *Patterns for Computer-Mediated Interaction*. John Wiley & Sons, Ltd., 2007.

23. Michael Vonrueden and Wolfgang Prinz. Distributed document contexts in cooperation systems. In Boicho N. Kokinov, Daniel C. Richardson, Thomas Roth-Berghofer, and Laure Vieu, editors, *Modeling and Using Context, 6th International and Interdisciplinary Conference, CONTEXT 2007*, LNCS 4635, pages 507–516. Springer-Verlag Berlin Heidelberg, 2007.

24. Weigang Wang and Jörg M. Haake. Tailoring groupware: The cooperative hypermedia approach. *Computer Supported Cooperative Work*, 9(1):123–146, 2000.