UNIVERSITÄT
DUISBURG
ESSEN

ABTEILUNG FÜR INFORMATIK UND
ANGEWANDTE KOGNITIONSWISSENSCHAFT

FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN

INFORMATIK

Technischer Bericht Nr. 2009-02

Context Modelling for Adaptive Collaboration

Jörg Haake, Tim Hussein, Björn Joop,
Stephan Lukosch, Dirk Veiel, Jürgen Ziegler

26.06.2009

ISSN 1863-8554

IMPRESSUM:

Technische Berichte der Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen

ISSN 1863-8554

Herausgeber:

Abteilung für Informatik und Angewandte Kognitionswissenschaft
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen
Campus Duisburg
47048 Duisburg

http://duepublico.uni-duisburg-essen.de/informatik/berichte.xml

CONTEXT MODELING FOR ADAPTIVE COLLABORATION

Jörg Haake[1], Tim Hussein[2], Björn Joop[2], Stephan Lukosch[3], Dirk Veiel[1], Jürgen Ziegler[2]

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

[2]Department of Computer Science and Applied Cognitive Science, Interactive Systems[a], University of Duisburg-Essen, Lotharstr. 65, 47057 Duisburg, Germany
{tim.hussein,bjoern.joop,juergen.ziegler}@uni-due.de

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

Collaborative work is characterized by frequently changing situations and corresponding demands for tool support and interaction behavior provided by the collaboration environment. Current approaches to address these changing demands include manual tailoring by end-users and automatic adaptation of single user tools or for individual users. Few systems use context as a basis for adapting collaborative work environments, mostly focusing on document recommendation and awareness provision. In this paper we present, firstly, a generic four layer framework for modeling context in a collaboration environment, secondly, a generic adaptation process translating user activity into state, deriving context for a given focus, and executing adaptation rules on this context, thirdly, a collaboration domain model for describing collaboration environments and collaborative situations, and, fourthly, examples of exploiting our approach to support context-based adaptation in four typical collaboration situations: co-location, co-access, co-recommendation, and co-dependency.

Keywords: context modeling; context awareness; adaptation; collaboration; CSCW

1. Introduction

Work in modern organizations is to a large extent collaborative. This is particularly true for knowledge work which is increasingly performed by distributed teams cooperating across large, often global distances. While collaboration has become ubiquitous, it also poses a number of challenges that need to be addressed for supporting it effectively by information and communication technology, or, more specifically, by cooperation support systems. These challenges arise from a variety of features that are characteristic for collaboration. Among other aspects, collaborative tasks are often ill-structured at the outset, emerge in the course of the collaborative process, and need to respond flexibly to changing goals or situations. Users participating in a collaborative project may find themselves in different physical environments or settings and may use a variety of different devices. Also, users are often involved in more than one project at a time, raising the

[a] The authors are in alphabetical order.

need for frequent task or tool switches and for rapid cognitive adjustments to the subject at hand.

Collaboration environments typically provide a range of features supporting the basic requirements of communication, coordination, collaboration (cf. Ref. 5) and various mechanisms for information provision and access[38]:

- An example for communication support is the provision of communication channels among co-workers on a shared document, like e.g. audio or video conferencing tools or text-based messaging systems.
- Examples for coordination support range from workflow mechanisms supporting the definition and execution of workplans to more informal approaches like the provision of awareness mechanisms for distributed teams collaborating in synchronous meetings, the provision of notifications about changes performed by team members during asynchronous collaboration in a shared document repository, or the provision of telepointers during synchronous joint editing of a document. Other means for facilitation affect the interaction among team members, e.g., paragraph locking in a shared editor prevents conflicting changes, or opening a simultaneous audio channel allows authors to discuss and coordinate their concurrent changes to the shared document (the latter being an example of using communication for coordination).
- Examples for collaboration support include the provision of shared editors or application sharing functionality which enables the team members to either synchronously or asynchronously work on a shared artifact in order to achieve a shared group goal.
- The provision of information pertaining to the subject of collaboration may concern any type of documented information ranging from artifacts produced by the project over relevant background knowledge, reference materials, web collections and others. Systems supporting these needs include, for example, shared workspaces, document management systems, wikis or shared bookmark collections. While many documents may be relevant for the team as a whole, projects typically comprise different roles and skills with different responsibilities and information needs. Users should therefore have efficient access to information relevant to their respective role in the project. At the same time, they should be able to get and maintain an overview and understanding of the growing pool of information used by the collaborators.

Collaboration support thus requires a wide range of communication facilities, tools and information resources that must be used in ever-changing collaboration situations in an effective and efficient manner. This often leads to a high level of complexity of the collaboration environment, to cognitive overload on the part of the users, inefficient interaction and, consequently, suboptimal use or outright rejection of sophisticated collaboration support systems[52]. To alleviate these problems, the functionality and interaction afforded by the collaboration environment should be adjusted to the changing collaboration situations. For example, when team members edit shared documents at different times (asynchronously) the environment could use notifications to facilitate coordination within the team. However, if several authors are online and access the same document, the environment should either switch to pessimistic concurrency control (e.g. locking) to prevent conflicts, or provide awareness features (e.g. telepointers and indications where coworkers are working) to help avoid conflicting edits. Similarly,

depending on the task and situation at hand, team members should be able to access relevant information as directly as possible, using, for example, shortcuts or optimized navigation paths. To date, when users and teams wish to adjust their environment to different situations, they have to negotiate and perform such changes manually. This leads to a high cognitive overhead, ignoring the potential for improvement, and subsequent suboptimal team performance.

The approach proposed in this paper is to address this problem by making the collaboration environment adaptive. Self-adaptation of systems to changing user needs and situations has been investigated for a long time in various application domains, such as intelligent tutoring systems, product recommendations in e-commerce, or location-based services (for an overview, see Ref. 33, for example). In these developments, the focus has mainly been on the individual, e.g. by implementing fixed or dynamically changing user models. In collaborative work, adaptation needs to take account both of the individual participant's needs as well as of the group's requirements. Adaptive activity support for collaborative settings has as yet been little investigated (maybe with the exception of adapting resources in physical meetings) and raises a number of new issues beyond single-user adaptation such as combining individual user models into group models, balancing the needs of different participants, and adapting a collaborative environment for improving group interaction.

Adaptive collaboration environments must utilize both information about the collaboration situation, and knowledge about adaptations suitable for the given situation. We refer to the information about the collaboration situation as "context information" while the second type of information will be called "adaptation knowledge". The notion of context is paramount for any kind of adaptive system. Context-aware and context-adaptive systems have in recent years been a major research topic in fields such as ubiquitous computing or mobile applications. Location-based services on mobile devices are one prominent and frequently cited example of context-aware systems. In accordance with the primary interests in ubiquitous computing research, context-awareness has often been associated with variables in the external physical environment such as location, proximity, time, or device used. This focus on physical context, however, seems to be too limited for adapting user-system interaction in general and, specifically, interaction in collaborative environments. A number of proposals have been made in recent years to consider a more general and comprehensive interpretation of context[18,32,51]. These proposals include also non-physical aspects such as the user's interests, tasks, or interaction behavior and may go as far as considering everything as potential context[6]. These definitions notwithstanding, a generalized view of context and methods for its' systematic use in adaptive interactive systems are still missing. This is even more pronouncedly the case for collaborative applications where context cannot only been seen from the perspective of the individual user but must be aggregated in a meaningful way for the group as a whole.

In this paper, we present an attempt to formulate a notion of context that is applicable for collaborative work. We elaborate a generalized model of context in conjunction with

a graph-based representation that is independent of the respective adaption mechanism. We also introduce a method for managing dynamic context in an adaptive system and present a system architecture suitable for adapting collaboration support. The model and methods are illustrated by examples of applying our approach in several collaborative situations. Finally, we present our conclusions and directions for future work.

2. Problem analysis and objectives

A basic methodological prerequisite for designing and implementing systems that are capable of adapting themselves is to gain an understanding and to provide an appropriate definition of the range of dynamic parameter changes that may cause a system to adapt. A considerable number of categorizations or multi-dimensional descriptions of such context-relevant parameters has been developed in related research (see e.g. Ref. 54). An account of these approaches will be presented in the related work section of this paper. The set of parameters actually considered for adaptation has typically been restricted to a few specific entities, such as fixed categories of users in static user models or the spatial coordinates of the user in location-based adaptation. Context-awareness of systems has frequently been associated, as in the latter example, with parameters that are external to the system under consideration. Depending on the definition of the system's boundaries, the external context may include or exclude aspects of the user. In ubiquitous computing research, user models were typically not characterized as context. From an interactive and collaborative systems perspective, it appears reasonable to consider the computational state of the hardware/software system as well as the state of the interaction with the user and among the users as internal context factors, and measurements taken from the physical environment as external. The user's 'state of mind', i.e., for example, goals, prior knowledge or interests, may initially be external, but becomes internal when represented in the system in some suitable form.

Since we are aiming at a generalized notion of context, we will consider both aspects as context and will refer to external parameters as exogenous context factors and to internal parameters as endogenous context factors. Whereas a number of approaches for context adaptation only take single or a few unconnected factors into account, we postulate that, in principle, any exogenous or endogenous parameter can become context. Whether some entity is context or not, is thus merely a matter of the perspective taken. This can be phrased in terms of focus versus context. Take the example of a typical location-based service: When setting the focus on 'restaurants and entertainment', the city the user is currently in becomes context. This may cause the system to show only restaurants and entertainment events in that particular city. Conversely, if the user wants to find all cities where a particular restaurant chain has locations, 'cities' may become a focus with the specific restaurant chain as context. Whether an entity is in the focus or constitutes context is determined by the current goals or activities of the actor or actors who interprets the context (cf. Ref. 51). Making use of this flexible, dynamic focus setting and the resulting relativity of context, however, leads to a strong interrelatedness of endogenous and exogenous factors and to a high level of complexity.

While many studies have addressed context-aware systems for individual users, the investigation of context for collaborating groups is as yet rather limited. The notion of shared context has been analyzed to some extent, for instance, in Ref. 54. However, how individually relevant context parameters can systematically be combined into a shared context is still a largely open issue. Approaches such as determining the overlap (intersection) between individual contexts or merging the different contexts (union) need to be investigated for different situations and activities. Such fairly manifest approaches, however, may be too limited and may need to be extended and complemented by more sophisticated methods.

To be able to translate these conceptual deliberations into adaptive collaboration support, we need a framework and methods for modeling context in an explicit, systematic way that is suitable for supporting the design and implementation process.

We are therefore aiming at a context definition that

- is sufficiently general to capture a wide range of adaptation purposes, in particular adapting both functionality and information content provided by the system in the current situation,
- can be expressed by formal representations that can be used as the basis for implementing or generating adaptive systems, and
- is capable of capturing the relevant context aspects specifically in collaborative work.

To exploit context in a collaborative system the context framework needs to include the following aspects:

- representing context in a formalized manner suitable for manipulation and interpretation in a collaborative adaptive system. Since developers may not foresee all adaptation requirements the context model needs to be extensible.
- representing adaptation knowledge that specifies the concrete adaptations to be performed in specific collaboration situation. Adaptations may address a range of different aspects such as tools, user interface and behavior, content, or even the context model itself.

For implementing and performing adaptations the following requirements must be met:

- the approach should facilitate and simplify the construction and implementation of context adaptive systems by providing means to separate the different concerns into manageable components,
- context model and application models must be consistent in terms of concepts and relationships as well as in terms of the representations used, and
- the adaptation knowledge must be consistent with the adaptation interface offered by the collaborative applications.

The approach proposed in this paper is to provide a generic, multi-layer context modeling framework, separating the different concerns in representing and utilizing context and providing a structure for the development process. This framework can be populated by different concrete modeling methods. The method we describe here uses ontological representations to model context by an extensible set of entities and relationships. We describe a neuronally-inspired spreading activation approach as well as rule-based techniques for contextualizing entities in this framework. We demonstrate the feasibility of the approach through different collaboration scenarios that comprise typical collaboration patterns.

3. Related Work

Context. The word *"context"* shows its meaning inherently: *con* (meaning: with) *text*. This definition has its origin in linguistics expressing the surrounding situation for an easier interpretation[36,4] or to express a communicative goal[7]. In philosophy context is either used as a correlation between sentences or defines aspects of a situation which help understanding the semantic meaning of an expression[9,29]. Psychologists research context with regard to how changes of the situation affect cognitive processes[17,53].

In recent years, the notion of context has played an increasingly important role in computer science, most notably in the area of ubiquitous computing with the aim of developing context-aware systems[45]. In the view of ubiquitous computing context-aware systems utilized contextual information such as time, location, users and available resources to represent aspects of the physical world. Roth[42] extends the set to include sensors and object properties. An even more general definition was provided by Dey et al.[18,19] defining context as any information characterizing the situation of an entity.

According to Zimmermann[54], any information used as context of an entity can be classified into one of five categories: individuality (all information about an entity that can be observed), activity (goals, tasks, actions), location, time and relations between entities. But this definition does not consider dynamic collaborative situations occurring in user groups or teams.

Another popular definition given by Winograd[51] defines context as an operational term for characterizing its role in communication. This means something is context because it is used in communication for interpretation and not due to its inherent properties. Edwards[24] explores the space between two different views on context: while the area of Computer Supported Cooperative Work (CSCW) considers people as consumers of context information; the ubiquitous computing community primarily considers systems as consumers of context information. Dourish[23] describes two main uses of context in ubiquitous computing: to either dynamically change the behavior of the system or to enrich information with retrieval cues. Either way, user models are typically not characterized as context.

Approaches to model context range from simple key-value models over graphical models up to sophisticated ontology-based which support validation and reasoning (cf.

Ref. 47). Recently, the use of the Web Ontology Language (OWL)[b] has gained importance (e.g. Ref. 10, 50). Other authors propose hierarchical context models: Chen and Kotz[12] distinguish directly sensed *low-level* context (time, location) from inferred *high-level* (mood, intention).

Dey et al.[18] call context information with a direct influence on the situation (like a participating person) primary context and complementary information without direct influence (like the person's phone number) secondary context.

Context-aware Systems. The most prominent examples for context-based adaptation focus on single users and consider location as the most relevant information[45,1,8], focus on learner profiles (e.g. the COLER-system[15]) or focus on adaptation of web content in regard to user interests. CATWALK[31] or SPREADR[30] are examples for adaptation of web content using OWL to model multiple domains which represent the context. CoBrA[10,11] is also using ontology-based context knowledge to adapt service agents according to a user's context. The Kimura system[35,48] is an example for supporting users through awareness information inferred from the working context of a user. It focuses on traditional office computing environments and uses peripheral displays to assist users in managing multiple tasks.

The aCAPella system[20] is a context aware system that can be "trained" by the user to automatically recognize events depending on the current context. Context information is obtained from microphones, cameras, RFID and other devices. aCAPella interprets the information and thus is able to recognize the start of a meeting and to automatically present documents that have been used in a similar context in the past. This approach focuses on real-world interaction and does not support collaborative work via computers.

Adapting collaborative systems. Gross and Prinz[26] introduce a context model and a collaborative system supporting context-adaptive awareness. The context model consists of events, artifacts, locations, etc. The main restriction of this approach is that the context representation can only be used to update and visualize awareness information. Additionally, only one cooperative application (BSCW) and no work environment with several applications was examined. Ahn et al.[2] introduce a knowledge context model used for implementing a virtual workgroup support system. The drawback of their solution is that their knowledge context model has to be extended for other application domains. The Semantic Workspace Organizer[39] is an extension of BSCW. It analyzes user activities and textual documents inside the shared workspace and suggests appropriate locations for new document uploads or for document searches.

Intermezzo[24] tries to fill the gap between CSCW and ubiquitous communities use contextual information through the creation of new higher-level services. Another approach was presented by Rittenbruch[41] to present context as awareness information - but real world examples are missing. Fuchs[25] describes a system called AREA which is an integrated synchronous and asynchronous notification service for awareness information. Again AREA uses the context representation only for awareness information.

[b] http://www.w3c.org/2004/OWL

The ECOSPACE project aims at providing an integrated collaborative work environment[37,40]. For that purpose, it 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 however has not yet been described[37,49].

The project inContext[21] focuses on enriching web services in mobile environments used for collaborative work. Schall et al.[44] introduces the Human-Provided Services (HPS) framework which lets people manage their interactions and integrates their capabilities into web-scale workflows as collaborative services. Dorn et al.[22] analyses team behavior and used collaboration services to improve adaptation of collaborative web services. The dynamic environment of a team in regard to activities and users is used as contextual information.

Summary. The above approaches focus on context representations and adaptations with limited scope. They are either restricted to single-user systems, to specific domains or to specific aspects of collaborative work, often focusing on awareness or knowledge management. Context is not represented in a formal manner and most context models are not extensible. Adaptation of general interaction capabilities 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 sufficiently support a context-based adaption of shared work environments.

4. A Generic Context Framework

In this section, we introduce a conceptual framework for context-adaptive applications that aims at meeting the requirements postulated in chapter 2.

4.1. *A Four Layer Framework for Context-based Adaptation*

In contrast to monolithic context models, we propose a context model distinguishing three distinct layers: knowledge layer, state layer, and contextualization layer. In addition, we add a fourth layer to our framework describing context-based adaptation. We see contextualization as a selection process: In a complex situation, contextualization mechanisms are used to extract the most relevant elements. Consequently, we need the following components for a conceptual context model:

(a) Information about the current situation/state provided by sensing components (mapping internal and external information sources into state objects).
(b) Background information about the (application) domain.
(c) Contextualization rules to constitute a contextualized state, and
(d) Adaptation rules that define a set of meaningful adaptations according to the contextualized state.

In order to meet the requirement of a separation of concerns (see section 2), the model is divided into distinct layers, including one layer for each the domain knowledge,

the current state, and contextualization as well as adaptation information. The separation into distinct layers helps to ensure the interchangeability of the components:

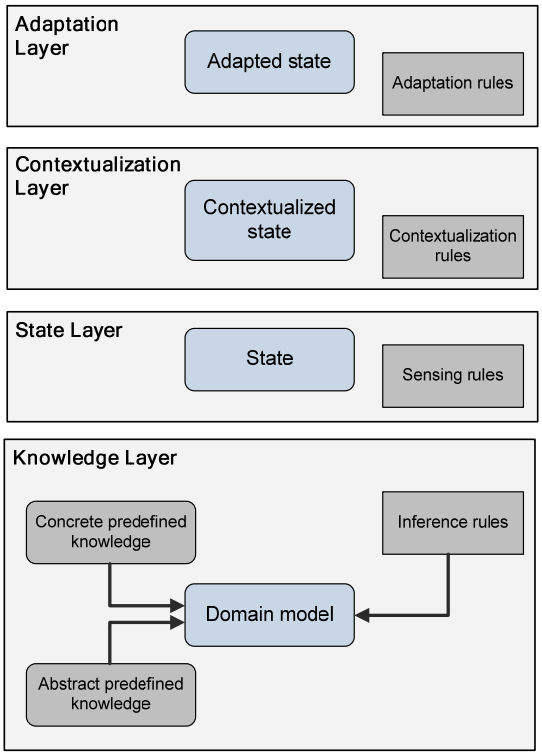

Figure 1 A four-layered framework for context-based adaptation.

1. The knowledge layer forms the basis for the context framework. It contains all relevant conceptual and factual knowledge about the application domain. It can be seen as a user- and situation-independent, neutral and objective view of the application domain, that includes all information about the system and application domain that is not likely to be changed.
2. The state layer contains information about the current situation including information about physical environment, computing environment, resources and user model: *Where is the user? What time is it? What are the current circumstances?* Considering the constitutional information defined in the knowledge layer, a model representing the current state of usage is being defined. Sensing rules express how both external and internal information sources can be used to take information from outside the application into account as well as information derived from system behavior (User A clicked on Item B). Thereby new objects or properties can be established under the terms defined in the domain model.

3. In the contextualization layer, contextualization rules define which subset of the state is relevant for a given focus. A focus can be an arbitrary subset of the state, which represents the adaptive system's current center of attention. This involves an interpretation of the state. What is important under which circumstances? An agenda for instance may generally be important in case of a business meeting. The result of this interpretation is a contextualized state holding all information relevant in a certain focus. We refer to this contextualized state as the context of the given focus. Contextualization can thus be seen as a filtering process to extract the most important elements from the current state.
4. Upon this, adaptations defined in the adaptation layer are selected. From a set of adaptation rules, the relevant rules are identified using the contextualized state. Such adaptation rules could be for instance *"If communication is needed for collaboration and all participants have a preferred communication channel in common, then open that channel for each participant in a session."* Applying the relevant rules leads to an adapted state, which needs to be reflected in the collaborative applications' presentation and behavior.

As an addition, evolutionary components could be integrated in this architecture on all of the four layers, but this is out of the scope of this paper.

4.1.1. *Knowledge layer*

The knowledge layer provides the foundations by representing both the application domain model as well as different context aspects such as physical environment, computing environment, resources and user model in a unified way. For this purpose, classes of individuals and relations of predefined relation types are used. We use instances of these types – which we denote as *individuals* – to express stable predefined knowledge such as *"Duisburg is a city"*. We call information regarding concrete individuals *concrete knowledge* in contrast to abstract knowledge concerning classes and their relations. Both concrete and abstract predefined knowledge is stored in this layer as well as inference rules regarding this knowledge. This information constitutes the domain model, which can be represented by different techniques, for example, by using the constructs of the Web Ontology Language (OWL).

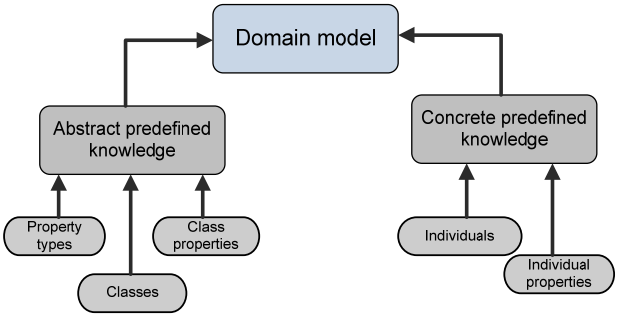

Figure 2 All relevant application and/or domain knowledge is stored in a domain model.

The following list briefly describes the most important concepts of OWL:

- *Classes:* Similar to common programming languages, classes represent categories or types of objects.
- *Individuals*: An instantiation of a class is called an individual. In object-oriented programming languages this would be called an object.
- *Literals*: Literals are simple values like strings or integer numbers.
- *Object properties*: A relation between two individuals is called object property (or object relation).
- *Datatype properties*: A relation that links an individual to a literal is called datatype property (or datatype relation).

Classes, individuals and literals can be represented by nodes while edges stand for object properties, datatype properties, or subclass relations. Figure 3 illustrates an example of concepts, permanent individuals and their properties.

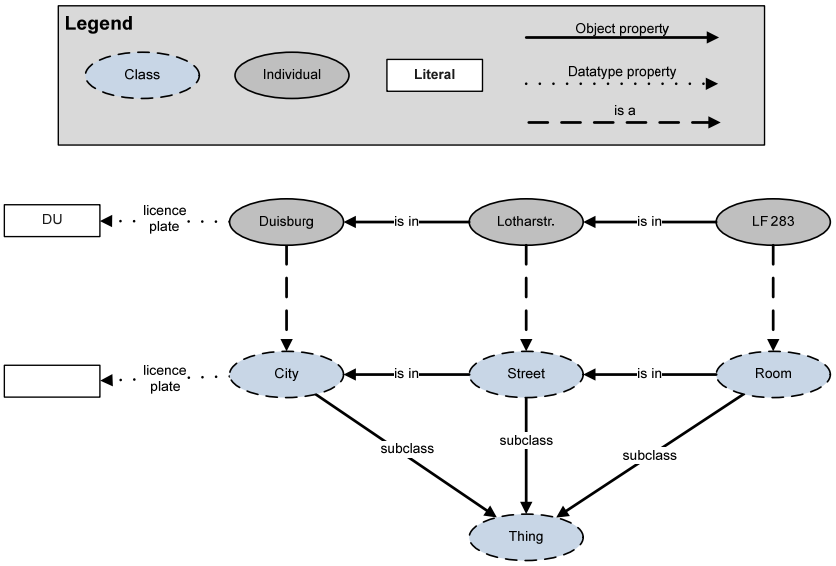

Figure 3 Sample domain model including classes, individuals, literals and properties.

Axiomatic knowledge can be represented in OWL as well: For each property type, domain and range of the property can be declared together with cardinality constraints, etc. A property type *works_together_with* could for instance be defined as a symmetric one, ranging from the set of Persons into the same set with an unbound cardinality.

Certainly, modeling the relevant information of the domain is a laborious task, but as depicted in section 2, a formalized representation of contextual information is a major

benefit in order to exploit context. Once created, a semantic model of the application domain can be utilized for various application and adaptation aspects, as we will show later on.

4.1.2. *State Layer*

Whereas the knowledge layer contains situation-independent information, the state layer represents the current state. Besides static individuals and properties, included in the knowledge layer, the state layer includes dynamically changing individuals and properties describing the current situation (state). However, these objects must conform to the pre-defined classes, axioms and property types of the knowledge layer. Sensing rules take information coming from endogenous and exogenous information sources (see section 4.2) and concrete knowledge from the domain model into account and reflect them in state individuals' and properties. Thereby sensing rules help to map and filter endogenous and exogenous observations of the environment (both external and system internal) into the state model. Abstract knowledge (*A person is in a room*) can be inferred from concrete state information (*Paul is in room LF283*) by applying inference rules from the domain model.

As a result, the application state holds a raw (in the sense of uninterpreted) model of the current state of the collaborative environment. To be more precise, we can express this state in a graph theoretical way introducing the term *state graph*:

Definition: State Graph

$G_S := \{V, E, \sigma\}$

The state graph G_S represents all individuals, classes (both represented by vertices $v \in V$) and properties (represented by edges $e \in E$) characterizing the current state. The property strength function σ expresses the "degree of truth" (a term originally from the field of Fuzzy Logic) and indicates the fulfillment of the property for each point of time in T. An uncertainty function for instance could be defined analogously if needed.

Definition: Relation strength function

$\sigma : E \times T \rightarrow [0,1]$

G_S represents the current state and can be precisely represented by an RDF graph.

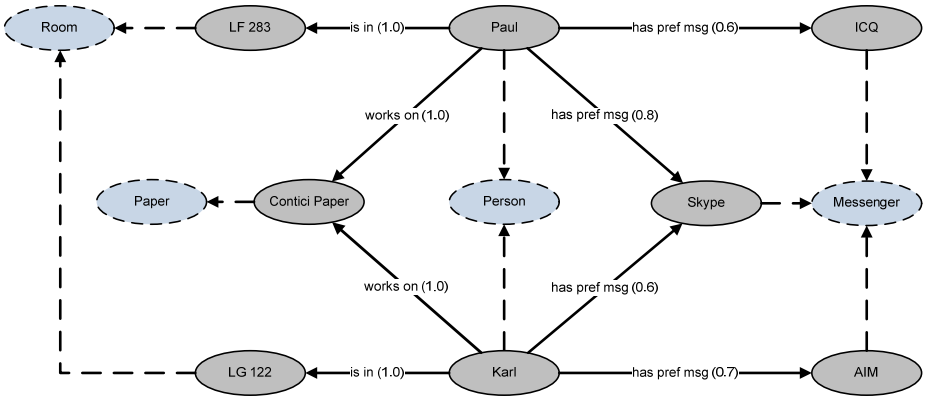

Figure 4 A state consisting of individuals (solid), classes (dashed) and properties (lines). The input comes from both sensors and domain model via inference (i.e. is a Person).

RDF assertions can be used to describe such a state graph in a machine readable way. In RDF, the relations are described in subject-predicate-object sentences.

Due to the separation of the layers, the constructed state model can be utilized for various purposes. For instance it can be used for contextualizing items as well as for analysis and learning processes.

4.1.3. *Contextualization Layer*

The contextualization layer provides techniques that define, which subset of the state is relevant for a given focus. Accordingly, we call those techniques *contextualization techniques* (e.g. rule-based). A focus is a non-empty set of objects from the state model (i.e. an arbitrary subset of the state) representing the current center of attention within the adaptive system. It is set by the application environment, e.g., by the user or by an application, and is used as the starting point for applying contextualization techniques. Contextualization techniques select those parts of the overall state that are contextually relevant for the current focus and then (optionally) apply some form of interpretation of that context e.g. by filtering or weighting the objects in the contextualized state or by inferring additional knowledge.

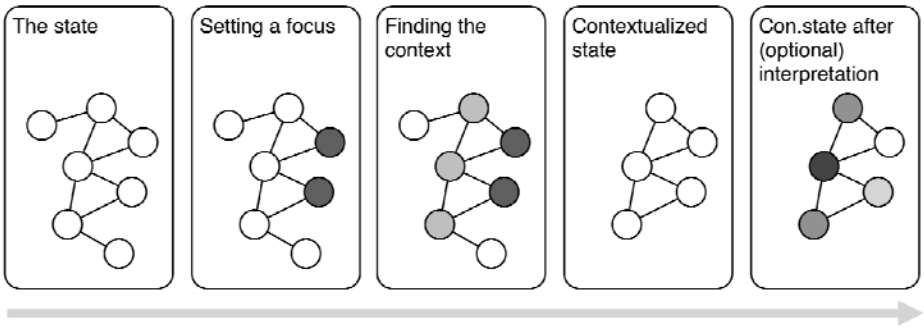

Figure 5 Step 1-4 illustrate the computation of the contextualized state as a filtering process. In an optional step, the contextualized state could then be interpreted leading to e.g. a weighted graph.

Next, we demonstrate this concept by giving several simple examples using IF-THEN rules as a contextualization technique (later on, we illustrate the contextualization process using alternative techniques). The IF-THEN-rules listed below indicate the importance of a class or individual under certain circumstances ("What is important under which circumstances?").

In case of a business meeting, an agenda is generally important. In a SPARQL-like notation this could be stated as:

Example rule 1: IF (?person participates_in ?business_meeting) AND (?agenda is_agenda_for ?business_meeting) THEN ?agenda

If two persons work on the same paper and have the same preferred communication channel, then this communication channel is important. We focus on the person and the business meeting and build the appropriate contextualized state by selecting objects (the agenda) contextually relevant to the focus objects.

Example rule 2: IF (?Person_1 works_on ?Project_Y) AND (?Person_2 works_on ?Project_Y) AND (?Person_1 has_pref_com_channel ?Com_c) AND (?Person_2 has_pref_com_channel ?Com_c) THEN ?Com_c

Here, the focus lies on people working on the same project. Applying this contextualization rule, we infer that they share a preferred communication channel, which is added to the contextualized state as well (because this seems to be an object of relevance in this certain context). Next, we give an example for a contextualization rule on a group basis:

Example rule 3: If (BSCW_Team ?predicate ?object) THEN BSCW_Resources

Under any circumstances, BSCW resources are important for the BSCW team. Translating this into the focus/context nomenclature, it means that we are looking at BSCW resources in any context as long as the BSCW team (focus) is involved. Consequently, BSCW resources will be added to the contextualized state any time, the BSCW team is involved.

A contextualized state can be seen as a filtered state. The result is a contextualized graph for each focus (e.g., user, group, artifact, or any combination of state objects), whereupon adaptations can be selected. Like the state layer below, separation of concerns and the use of a formal semantic representation like RDF make the contextualized state highly utilizable and interchangeable between applications.

Incorporating the relation strength values inside the state is optional and depends on the contextualization technique. In section 6.3, we describe a contextualization process using a Spreading Activation approach that uses the focus as a starting point for an *activation flow* resulting in a weighted network representing the corresponding context. Opposite to the IF-THEN rules Spreading Activation would incorporate the relation strength function. Alternatively, traditional recommendation algorithms like Collaborative Filtering could be used. Those techniques are triggered by a certain input (this would be the *focus*) and thereupon compute the most important individuals or classes by building a *context* around this input.

4.1.4. *Adaptation Layer*

While the contextualized state contains information about individuals relevant for the current focus, the adaptation layer contains adaptation rules, which describe which adaptation actions to perform in which case. Adaptation rules usually include operations for changing properties of the collaborative environment, of artifacts, or in general any state variable (“show”, “start application”, etc.). The effects of these adaptations have to be propagated to the user interface. We propose to represent adaptations rules as IF-THEN-rules. An example for such a rule may be: *IF ?document is important THEN show ?document*. Detailed examples for adaptations are presented in section 6.

4.2. *Utilizing the framework for adaptation*

Having defined and described the layers of our proposed context framework, we are now going to illustrate how the models and rules can be used in an adaptation process. Use-cases for validation of the framework and a prototype system based on it will be presented in sections 6 and 7.

An adaptation cycle usually consists of the following phases: User action – sensing – selecting adaptations – performing adaptations. Subsequently, we are going to show that the separation of models and rules proposed in the framework matches well the phases of such an adaptation cycle. Figure 6 shows a sample adaptation process based on the framework.

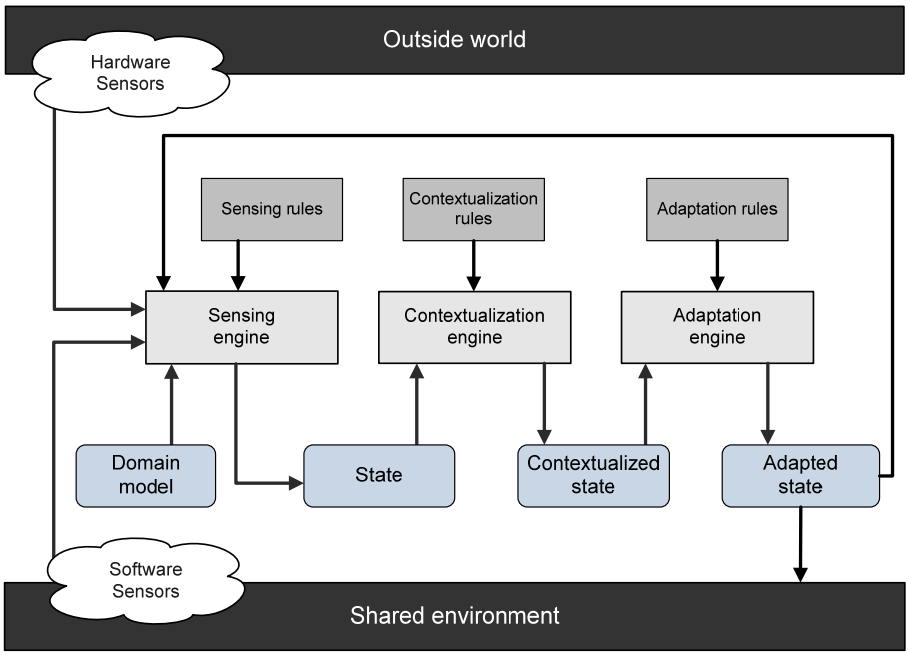

Figure 6 Adaptation process

4.2.1. *Sensing*

Sensor components monitor the users' behavior, ideally in an unobtrusive way. These observations may include events or conditions in the computing environment and from the outside world as well, including sophisticated methods that sense rather abstract information like *"Two persons, who are working on the same project, are in the room at the same time"* via sensing rules. A sensing engine processes this information and thereupon creates or modifies the state model.

4.2.2. *Contextualization*

Now the contextualization engine applies the relevant contextualization rules mapping the current collaboration state to a contextualized state. Such an engine could, for instance, be simply rule based or use sophisticated techniques like Spreading Activation for deriving the current context state. In section 4.1.3, contextualization rules are defined; section 6.3 provides an example for Spreading Activation. There are no constraints regarding the contextualization technique used as long as it provides means for selecting the most relevant elements.

4.2.3. *Adaptation*

At this point, the most important individuals and classes are identified as contextualized state and can be used as a foundation to perform adaptations. If a certain communication channel is important, the system can establish it for all participants. In the same way,

important documents can be presented. Hardware adaptations are also possible, like turning on an LCD projector or switching of the light. During this process, certain adaptation rules are triggered (depending on the contextualized state), an execution sequence is created and processed in the adaptation engine. This leads to updates of the collaboration environment as well as of the state model. Adaptation processes are illustrated in detail in section 6.

4.3. *Summary*

In section 2, we identified several major requirements for a generic context framework. Our approach helps to address these requirements through the following aspects.

Our definition of context is consistent with commonly cited definitions by Schilit & Theimer, Dey and others. Context is constituted by the entities that characterize the situation. By using contextualization rules, we identify just those entities and optionally use techniques such as Spreading Activation to refine the results. We interpret contextualization as an intelligent filtering process that distinguishes important information from unimportant. The preprocessing performed in the contextualization layer may help to reduce the complexity of the adaptation rules by simplifying the conditions to be checked. The layering and the specification of different components facilitate the flexible substitution of components in our architecture. Separating the aspects of sensing, contextualization, and adaptation allows substituting or combining the components implementing these aspects. For instance, it is possible to use different adaptation engines within a collaborative system on the same contextualized state.

The use of established semantic languages like RDF, OWL and SPARQL throughout the framework additionally supports interchangeability of components and “trading” of context models between applications. The approach is based on a unified formal graph model for context using different representations such as RDF, OWL or SPARQL. These representations can be interpreted and manipulated in a computer system. The adaptation rules contained in the adaptation layer facilitate the description of adaptation knowledge including the effects of adaptations. Our approach supports consistency between context model and application model due to the integrated knowledge layer representation.

In sections 5 and 6, we give examples of the framework’s universality and extensibility by sketching a sample domain model for collaborative workspaces and by applying the framework to the adaptation in typical collaborative situations. Further on, we propose a system architecture for this framework in section 7 to demonstrate its general technical practicability.

5. A Sample Domain Model for Collaborative Workspaces

The previous section introduced our generic context framework which is based on four layers: knowledge, state, contextualization and adaptation. The knowledge layer forms the basis for the context framework as it contains all relevant conceptual and factual knowledge about the application domain. Since this paper deals with context modeling for adaptive collaboration, the knowledge layer needs to describe knowledge about the

collaboration domain, the application domain (i.e. the task that should be collaboratively solved) as well as external context factors. In this section, we focus on the collaboration domain by analyzing properties of collaboration and its support in collaborative workspaces. This domain model addresses scenarios in which several actors collaborate to achieve a shared goal. Such collaboration is technologically supported by a wide range of communication facilities, tools and information resources. The collaboration process and result are documented in shared artifacts which are accessed by tools and thereby the collaborating actors.

The following model intends to capture the basic concepts of collaborative workspaces. It focuses on the technological support for collaborative interaction and does not distinguish different artifact types or task domains. If applied in a certain collaboration environment (e.g., BSCW, SharePoint), it must be extended with concepts matching its specific properties. Thus, the domain model is intended to be completely open for extensions that cover aspects not included in the following example scenario of two knowledge workers collaborating at a distance. Such extensions could stem from scenarios focusing on co-located collaboration, workflow management, collaborative recommendation or social networking. We address some of these topics in Section 6 where we extend the domain model presented in this section accordingly.

As discussed in Section 2, the collaboration and task domains are interdependent: knowledge about collaborative work (e.g., shared work environments and their components, roles of users, tasks) refers to concepts from the application domain (e.g., types of artifacts). Specific tasks or objects from the application domain may be worked on best by using specific types of collaboration support (e.g., types of sessions, tools, or coordination mechanisms).

5.1. *Global collaboration space*

To derive a domain model for collaboration, we start by analyzing and decomposing the basic concepts and interactions in a global collaboration space. A global collaboration space (cf. Figure 7) contains all actors, user workspaces, applications, services and artifacts needed to carry out a project between the involved actors. In our sample environment *Actor:a_alice* and *Actor:a_bob* are members of a team represented as *Team:productDesign* (cf. Figure 7) that uses several applications to design and describe a new product. The class *Role* defines a set of possible actions that actors can execute within an application. Each application defines a set of supported actions. A subset or the whole set of these possible actions is assigned to a role. Thereby, different roles with different access rights can be defined. Each actor can have multiple roles. The role of a user for an application is assigned by the user's workspace and needs to be explicitly modeled for each type of application.

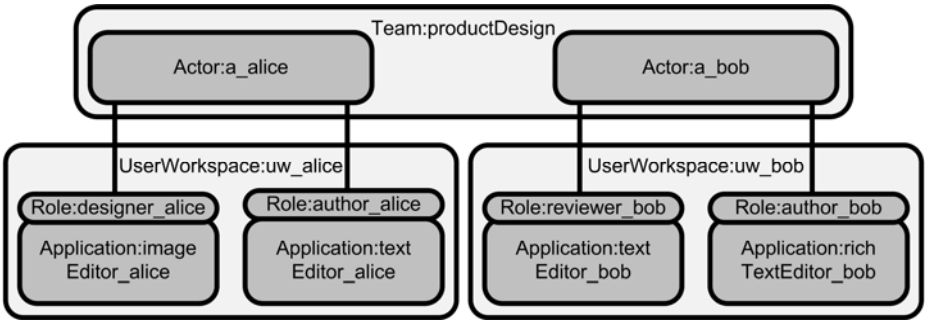

Figure 7 Basic interactions within a sample global collaboration space

As Figure 7 shows, *Actor:a_alice* has the role *Role:designer_alice* while using *Application:imageEditor_alice* and the role *Role:editor_alice* while using *Application:textEditor_alice.* We distinguish between these two roles to illustrate that each role usually implies different activities and uses domain specific applications to operate on artifacts. *Actor:a_bob* uses the *Application:textEditor_bob* and has the role *Role:reviewer_bob*. This role implies a restricted action set that is available to operate on artifacts. Usually, reviewers add comments to the text and reference other artifacts for supporting their claim. But, reviewers are not allowed to edit or delete text. While using the *Application:richTextEditor_bob*, *Actor:a_bob* has the role *Role:author_bob*.

5.2. *User workspaces*

Within a global collaboration space each user has its own user workspace: *UserWorkspace:uw_alice* and *UserWorkspace:uw_bob* (cf. Figure 7). As each user's workspace can be configured differently, a user workspace defines a set of available applications. Each of the two actors (*Actor:a_alice* and *Actor:a_bob*) interacts with two applications that reside in the corresponding user workspace. As soon as an individual, e.g. *Artifact:textDocument,* is accessed by more than one user, it is used collaboratively. Figure 8 shows the case of a collaborative text editor application with two individuals *Application:textEditor_alice* and *Application:textEditor_bob* accessing the shared individual *Artifact:textDocument*. Actions, e.g. *AddShape:as_alice* or *OpenText:ot_bob*, are used to describe an interaction between actors and the applications. When interacting with an application, actors perform actions which the application is capable of and which are allowed by their role. An application uses services, e.g. *Service:lineTo_alice* or *Service:setContent_bob*, to operate on artifacts, e.g. *Artifact:image* or *Artifact:textDocument*. Artifacts may be extended with coordination specific data and services, e.g. locking information.

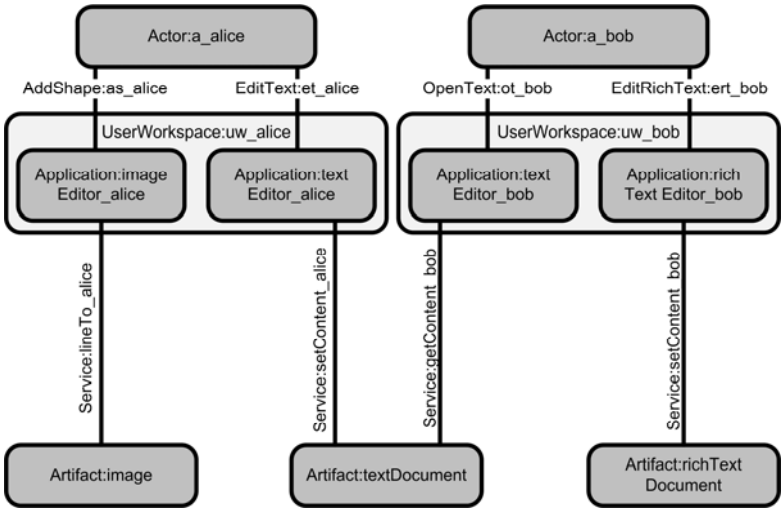

Figure 8 Basic interactions between user workspaces and shared model

5.3. *Collaborative applications*

The previous section decomposed a global collaboration space into user workspaces. These user workspaces offer applications to their users which in the sense of service-oriented architectures can be further decomposed. For that purpose, we consider that collaborative applications implement the model-view-controller (MVC) paradigm[34] (cf. Figure 9). Actors interact with the application by performing actions allowed by their roles. These actions are received by the corresponding controller components of the application. In Figure 9, *Actor:a_alice* performs an *EditText:et_alice*. This action is received by the *Controller:textDocument_alice*. The controller then uses the *Service:setContent_alice* to modify the *Artifact:textDocument*. As in MVC, the shared model then uses the *Service:notifier_alice* and *Service:notifier_bob* to notify registered view components, e.g. *View:textDocument_alice*, about changes of the model. When receiving such a notification the registered view component can update the display.

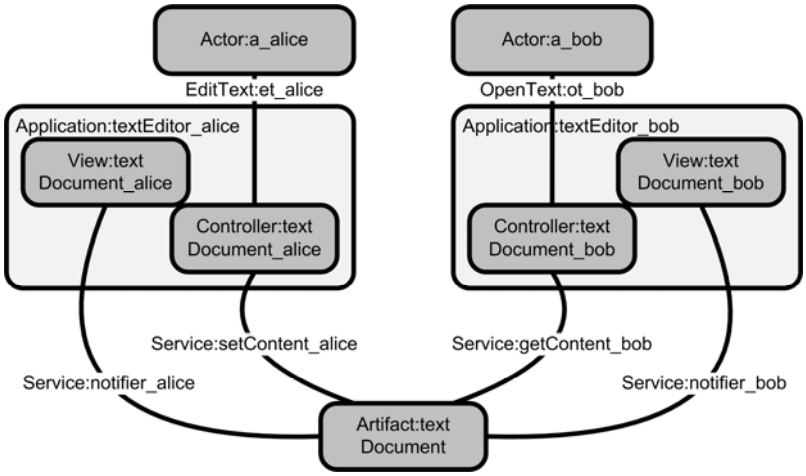

Figure 9 Decomposition of collaborative applications using the model-view-controller pattern

5.4. *Summary*

After decomposing and analyzing a global collaboration space, Figure 10 summarizes the domain model for collaboration and shows the basic classes and their relations that can be used to describe collaboration context in a global collaboration space. In addition to the above classes, we introduce the class *ApplicationFunctionality* used to express the functionality an *Application* offers and the class *ApplicationFactory* describing for each workspace which applications are available.

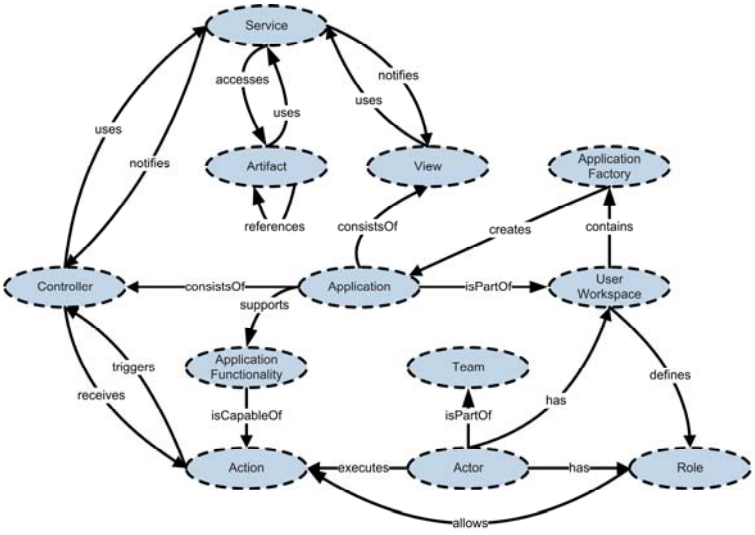

Figure 10 Domain model for collaboration in a shared workspace

The *ApplicationFunctionality* class has several subclasses not shown in Figure 10, e.g., *Communication*, *SharedEditing*, *Awareness*, *Management* or *WorkflowManagement*. All of these classes distinguish further subclasses. The *Communication* concept, e.g., distinguishes between *Synchronous* and *AsynchronousCommunication*. The class *SynchronousCommunication* then distinguishes between *Audio*, *Video*, or *Chat*. Similarly, the *Awareness* class distinguishes between *Synchronous* and *Asynchronous Awareness*. The class *SynchronousAwareness* then distinguishes between e.g., *ActiveNeighbors*, *ActivityIndicator*, *RemoteFieldOfVision*, *RemoteCursor*, *Telepointer*, or *UserList*. The *Management* class, e.g., distinguishes functionality for *AccessRight*, *Session*, *User*, or *ConcurrencyControlManagement*. The *SharedEditing* class distinguishes different kinds of editors for, e.g., *Text*, *RichText*, *Image* or *Calendar*. Most of these classes are derived from patterns for computer-mediated interaction[46] which describe best practices for designing tools for collaboration.

Figure 11 Application functionality concepts and relations

Figure 11 shows an excerpt of this domain model class hierarchy highlighting the *Chat* application functionality. As result, each application which supports chat application functionality has to offer at least two action types: *OpenChat* and *SendMsg*.

All above classes are useful to model the configuration of shared workspaces and tools and to capture the current context in the state layer at runtime. In the following section, we show how adaption rules based on the above domain model are able to recognize specific collaboration situations and trigger adaption.

6. Applying the approach in collaboration situations

In order to assess the applicability of our approach we discuss its application in four typical collaboration situations:

- Co-location denotes the situation where several people meet at a physical (e.g. meeting room) or virtual location (e.g. shared artifact or shared meeting space),
- Co-access denotes the situation where several people access the same artifact (e.g. a design drawing or document),
- Co-recommendation denotes the situation where explicit or implicit actions of the collaborators are used to suggest information resources potentially useful for a common task, and
- Co-dependency denotes the situation where several tasks, objects or users are dependent (e.g. tasks are dependent on tasks being worked on by other users).

We argue that these situations are typical for many collaboration scenarios or episodes. For example, while working on projects members meet regularly (co-location), access and manipulate joint artifacts (co-access), point each other to relevant information during project work (co-recommendation), and work on dependent tasks or artifacts such as related documents (co-dependency).

In the following, we will discuss how our approach can be used in each of these situations. For each situation we will present a short scenario illustrating the situation, present a relevant part of the domain model (extending the basic domain model presented in section 4 and 5), show a relevant excerpt of the state model of the current situation, and discuss how this state model can be contextualized (i.e. filtered to those parts relevant for making adaptation decisions). We will then show examples of adaptation rules illustrating how the contextualized state model can be used to decide on adaptations, and how the adaptation itself is in turn reflected in an adapted state model.

The aim of these scenarios is to provide evidence for the applicability and validity of our approach.

6.1. *Co-Location*

Co-location occurs when two or more people, artifacts or devices are physically near to each other. Co-location affords a range of adaptations, for example, facilitating the use of nearby devices or automatically setting up the collaboration system for joint tasks or information access.

Scenario: Alice and Bob, jointly working on a report, gather in a meeting room equipped with a large shareable display. Sensors recognize and identify the two persons. The system infers that they will likely work on the current report. It activates the display and shows the report in the workspace. Alice and Bob can now immediately start discussing the latest version.

For this scenario, we extend the domain model shown in Figure 10 by a class for locations and a class for devices. An actor works with a device, such as a computer or a display, so the device must be able to display artifacts. Additionally, each actor and each device can be at some location. Actors who are at the same location are linked by an

additional relation *isLocatedWith* (see Figure 11). This relation is instantiated by sensing rules as described in 4.1.2.

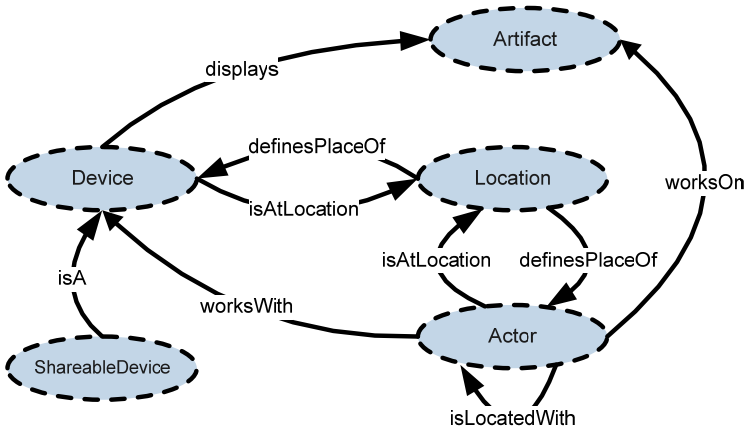

Figure 12 Excerpt from Figure 10 extending the domain model with classes for *"Location"* and *"Device"*

The state is contextualized by rules (cf 4.1.3) expressing which concepts or individuals are important for a given focus in this model. In the example, a rule could, for instance, state: “IF at least two persons are in the same room AND work on the same artifact AND the room has a shareable device (e.g. a large display) THEN this particular artifact and device are important.” In this case, persons, shared artifacts and locations are the focus points, for which potentially important shared artifacts and matching shareable displays are computed as context. We define the context by examining state objects reachable from the focus using three conditions: Is the artifact simultaneously being worked on? Are the persons working on it in the same room? Does the room have a shareable display? This contextualization rule could be described in SPARQL as follows:

```
CONSTRUCT {?sub ?pred ?obj}
WHERE {
   ?actor1 hasLocation ?location .
   ?actor2 hasLocation ?location .
   ?actor1 worksOn ?artifact .
   ?actor2 worksOn ?artifact .
   ?location definesPlaceOf ?shareableDisplay .
}
```

This query CONSTRUCTs the context for a certain focus: The WHERE-part stands for the focus, from which we define relevant artifacts and devices. CONSTRUCT extracts a submodel from the state that fulfils the statements inside the WHERE-part. Optionally, the subgraph could be constrained to assertions matching a predefined template by using concrete values instead of the wildcards ?sub ?pred ?obj (= any

subjects, predicates, and objects allowed). The contextualized state is shown in Figure 12. In our scenario, it includes the location *Location:l_meetingRoom* where both *Actor:a_alice* and *Actor:a_bob* are now and the devices at that location (here: *Device:d_shareableDisplay*). Furthermore, all artifacts (Artifact:a_report) on which both actors currently work on collaboratively are included.

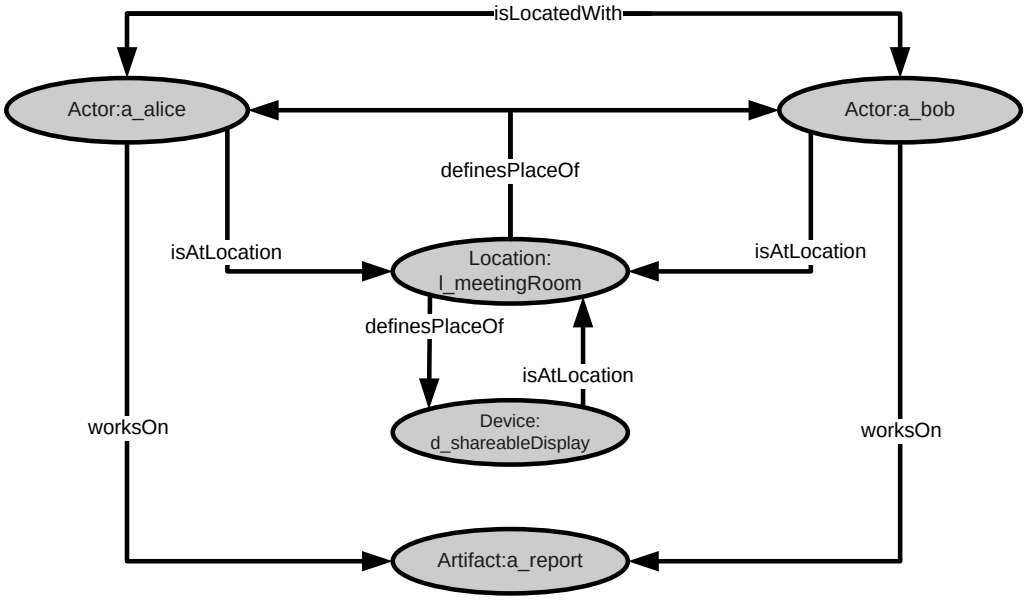

Figure 13 Contextualized state after creating a co-located situation

An adaptation rule, which adapts all devices in the room based on the contextualized state, could be formulated as follows:

```
// Obtain artifacts and device from the contextualized
// state
artifacts := getArtifactsFromContextualizedState;
device := getDeviceFromContexutalizedState;

// If there are shared artifacts and a shareable device,
// then display the artifacts on the display
IF (notEmpty(artifacts) AND notEmpty(device)) THEN {
  display(artifacts,device);
}
```

The adaptation rule is triggered when all parts of the condition are fulfilled: The adaptation rule is triggered, because *Actor:a_bob* enters *Location:l_meetingRoom* which *Actor:a_alice* has entered before. Both work on several artifacts simultanously and *Location:l_meetingRoom* provides a shareable display. Consequently, the adaptation rule obtains an artifact and a shareable display from the contextualized state (neither of the sets is empty) and *Device:d_sharableDisplay* will be activated and it will display *Artifact:a_report*.

6.2. *Co-Access*

Co-access denotes the situation where several people access the same artifact. 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 local workspaces. Alice then created a shared text document and opened a shared text editor to work on the design. Bob later opened the same shared text document to review the current state of the design. Both team members have different roles defining their tasks within the team. Both can use a chat application, though none is currently in use.

For this scenario we use the domain model shown in Figure 10. Figure 14 shows the state representing the above collaboration situation. For space reasons we only show context individuals and relations relevant for the adaptation discussed below.

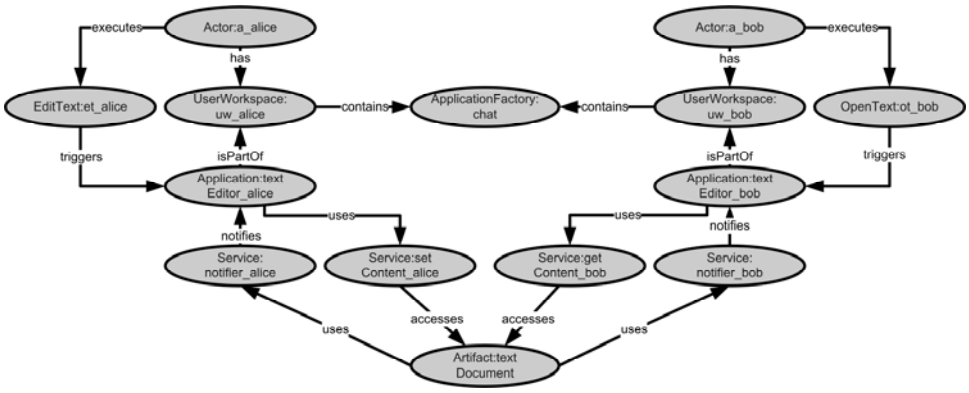

Figure 14 Sample state for co-access scenario

We assume that the team members can synchronously access and edit document parts, which can lead to conflicts or opportunities for collaboration. Different adaptation possibilities exist to improve the interaction within a team, 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.

In Figure 14 two actors access the same *Artifact:textDocument* and thus are part of the context of the artifact as well as of each other. Assume that *Actor:a_bob*, in the role of a reviewer, might want to ask questions for clarification for which he has to directly

contact an author of the document. Given the state described in Figure 14, establishing a communication possibility among the two actors is one meaningful adaptation possibility.

The tools which are available to all users of a team can be derived from the current context. In the state shown in Figure 14, both user workspaces, *UserWorkspace:uw_alice* and *UserWorkspace::uw_bob*, are connected to an *ApplicationFactory:chat* (i.e. both users are able to use a chat tool though none is opened yet). Thus, in the current state, *Actor:a_alice* and *Actor:a_bob* can both communicate via an *Application:chat*. The following pseudo code shows a contextualization rule (denoted contextualization block) and an adaptation rule which makes use of the contextualized state (cf. Figure 15) created through the execution of the contextualization block and adapts the users' workspaces accordingly:

```
// Contextualization block, ${focus} = OpenText
artifacts := getArtifactsInContext(${focus});
actors := getActorsInContext(artifacts);
communication := getApplicationsInContext(actors,
                 “Communication”);
// Adaptation rule
IF (isNotEmpty(communication)) THEN {
  // Select a communication tool and
  // open it for all actors.
  selectedApplication := selectOneFrom(communication);
  openForAll(selectedApplication,actors);
}
```

In the contextualization block, we define three successive filtering functions which extract the contextualized state from the state graph: `getArtifactsInContext`, `getActorsInContext`, and `getApplicationsInContext`. Below, we exemplify the definition of contextualization rules by presenting a SPARQL query implementing `getArtifactsInContext(${focus})`, where `${focus}` denotes the actual content of the focus variable maintained by the execution environment. If the current focus conatins several state objects, the execution function may simply execute the SPARQL query for each element of the focus set and add the results to the contextualized state graph. In our example, we apply the following contextualization rule to the *State* with `${focus}` being replaced with the current focus, i.e. the action type *OpenText*.

```
CONSTRUCT { ?sub ?pred ?obj }
WHERE {
  ?action triggers ?controller .
  ?action type ${focus} .
  ?controller uses ?service .
```

```
  ?service accesses ?artifact
}
```

For space reasons, we omit the contextualization rules used for the other two filtering functions in the contextualization block. The resulting contextualized state is shown in Figure 14.

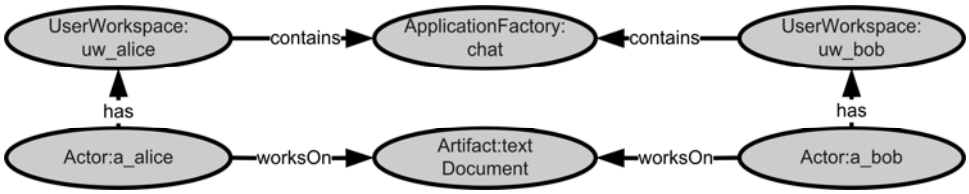

Figure 15 Contextualized state for co-access scenario

In our scenario, the above adaptation rule is triggered by *Actor:a_bob* opening the *Artifact:textDocument*. The function `getArtifactsInContext` returns a set of artifacts which are in the context of the action *OpenText:ot_bob*. The function `getActorsInContext` then calculates all actors which access the artifacts in the context of *OpenText:ot_bob*, i.e. in Figure 14 *Actor:a_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 14 *ApplicationFactory: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. These values are updated via a special learning algorithm. Here, a *Chat* application will be opened for Alice and Bob, as a *Chat* application is the only application available for *Communication* within both users' workspaces. The result of this adaptation is shown in the adapted state in Figure 16. Now, both actors use an *Application:chat* and collaboratively access an *Artifact:chatContent*.

As already mentioned above, different adaptation possibilities may exist if the *Application:textEditor* supports the corresponding *ApplicationFunctionality*. Imagine that the *Application:textEditor* in Figure 14 supports different possibilities for achieving awareness in synchronous interaction by, e.g., a *UserList*, *RemoteCursor*, *RemoteFieldOfVision*, and an *ActivityIndicator*. This would be reflected in the corresponding context graph as relations between the *Application:textEditor* and the corresponding *ApplicationFunctionality* concepts. Not all of the awareness widgets need to be used to reduce the cognitive load of the actors. In some cases, even all awareness widgets might be disabled (e.g., when different actors work on completely different parts of the document). By using the information stored in the context graph an adaptation rule can enable or disable specific awareness components in specific situations to improve interaction.

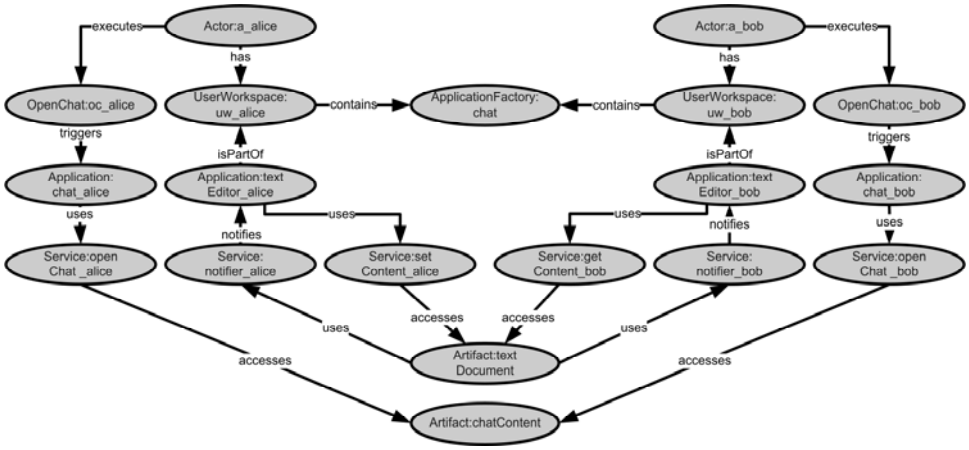

Figure 16 Sample adapted state for co-access scenario after adaptation

The following adaptation rule is an example for this: it enables one synchronous awareness widget (e.g., according to the preferences of the collaborating actors) when at least one additional actor accesses an artifact in the context of *OpenText:ot_bob*:

```
// Contextualization block, ${focus} = OpenText
artifacts := getArtifactsInContext(${focus});
actors := getActorsInContext(artifacts);
applications := getApplicationsInContext(actors,artifacts);
// Adaptation rule
IF (isNotEmpty(applications)) THEN {
  FOREACH (application: applications) DO {
    awareness := getFunctionalityInContext(application,
                 “SynchronousAwareness”);
    // Select an awareness widget and
    // open it for all members.
    selectedAwareness := selectOneFrom(awareness);
    openForAll(selectedAwareness,actors);
  }
}
```

Finally, let us consider an adaptation enabling a concurrency control mechanism. In this case, the *Application:textEditor* would have to support the corresponding application functionalities concerning concurrency control management, e.g. *OptimisticConcurrencyControl*, *PessimisticConcurrencyControl*, or *OperationalTransformation*. An adaptation rule, triggered by *OpenText:ot_bob*, would again first check whether other actors access the artifacts in the context of *OpenText:ot_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.

6.3. *Co-Recommendation*

Co-recommendation is a technique by which the system suggests potentially relevant information resources based on prior actions, e.g., ratings, of other group members and one's own interest profile. Large-scale recommender systems are in use in areas such as electronic business. Exploiting implicit or explicit relevance ratings of other users, however, has also a significant potential for collaborative work. In the following, we will show how the task of document recommendation can be supported by extending the respective part of the domain model, and by defining appropriate contextualization and adaptation rules. We also show how spreading activation can be used as a technique for performing contextualization.

Scenario: Bob works for a building company specializing in refurbishments. He is an expert in the area of energy efficiency and frequently tags documents or web pages with keywords from this domain. On this basis, the system classifies documents into a predefined semantic model. Alice is a new colleague and enters the system to search for information regarding thermal insulation. The system recommends a number of documents concerning this topic which have been frequently accessed by other colleagues.

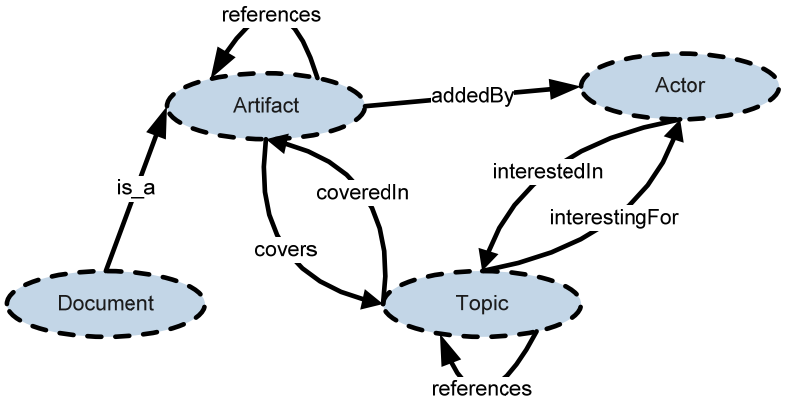

Figure 17 Excerpt from Figure 10 extending the domain model for co-recommendation.

Later, Bob adds a new document to the system which he rates as relevant for the topic roof insulation. Based on Bob's relevance rating, the system recommends this document to Alice immediately through an appropriate awareness function or when Alice logs in the next time.

In order to allow the system to recommend Alice documents, we need to extend the domain model from Figure 17 by including classes for "Topic" and "Document" and

create additional relations connecting them. We use a spreading activation algorithm (described later) to contextualize the state shown in Figure 18.

Each artifact has one or more topics assigned to it and may reference other artifacts as well. The relation strength represents the degree of 'relatedness' of two instances. For example, *Actor:a_alice* is very interested in the topic *Topic:t_energyInsulations* (represented by a value of 0.9). On the other hand, *Topic:t_energyInsulations* is only peripherally covered in the document *Artifact:a_manualProofing* – which is a manual for do-it-yourself house proofing –(relation strength is 0.4).

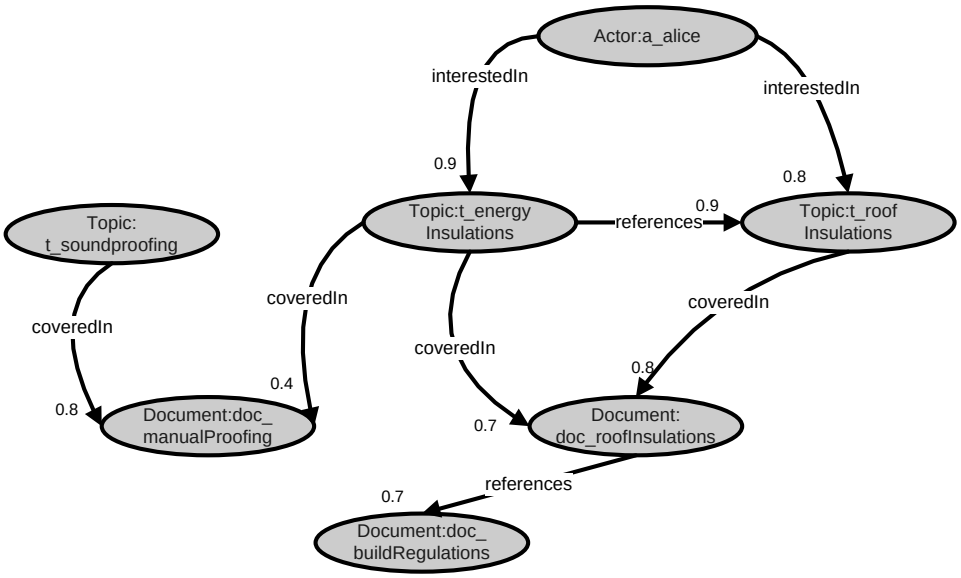

Figure 18 A possible state for co-recommendation scenario. Relation strengths are assigned to each relation (cf. section 4.1.2). Note that not all relations from the domain model in Figure 17 are included.

In this example, a spreading activation technique is used for contextualizing the content part of the state. The basic idea is to initially inject activation energy into one or more initial nodes and then propagate the induced energy through adjacent nodes through the network with decreasing charge. As a result, the elements with the highest activation can be selected as the contextually most important ones. The new activation values of the adjacent nodes j are computed as $A_j = A_{i+} \sum_i A_i w_{ij}$ where w_{ij} is the relation strength connecting node i and j. The spreading process from the newly activated nodes is continued until a predefined stop condition is met (i.e. a distance constraint or number of spreading steps).

The following algorithm for spreading activation can be used:

1. initialize the graph setting all activation values to 0.
2. select one or more initial nodes and set them to an initial value (e.g. 1.0)

3. for each edge e_{ij} connecting node *i* and node *j*, spread the activation to adjacent nodes
4. Once a node has propagated its activation mark it as processed
5. stop the spreading if a predefined condition is met

Tracing back to the ideas of Collins and Loftus[14] and Anderson[3] Spreading Activation techniques have particularly been used in information retrieval systems (see Ref. 12, 16 and 43 for further details).

For our scenario, a possible contextualization by spreading activation could work as follows:

1. Select the node *Actor:a_alice* as initial node (focus, cf. section 4.1.3) with A_i = 1.0
2. activate all vertices connecting *Actor:a_alice* to adjacent ones
3. propagate the activations until either:
   - the distance from the initial node to the current one is greater than 2
   - the relation strength is greater than 0.5
   - the propagating vertices' activation is smaller than 0.5

Figure 19 shows the result of the spreading activation algorithm applied to the graph shown in Figure 18. Note, that activations added at a later step will not be further propagated in the above two spreading activation algorithms.

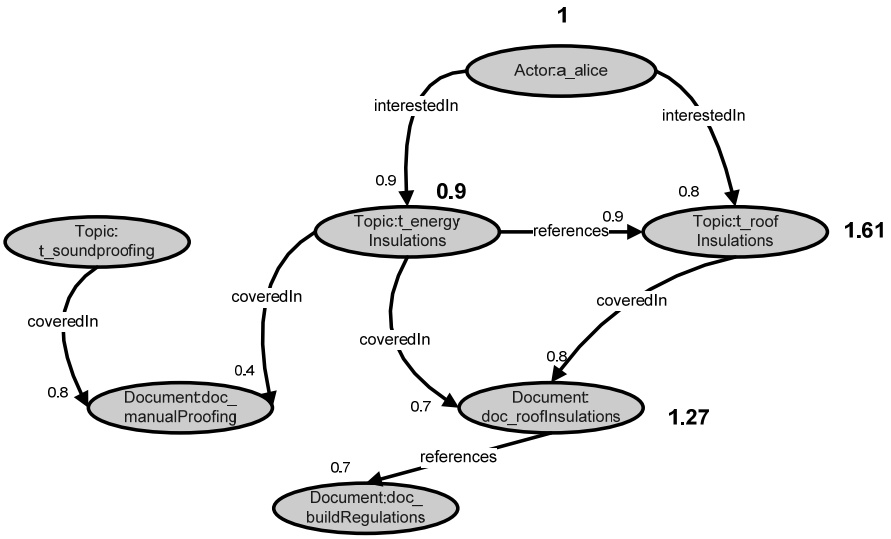

Figure 19 Contextualized state after spreading activation on the graph shown in Figure 18. Activations are shown bold highlighted.

The interpreted state shown in Figure 20 is the result of applying the spreading activation techniques described above and taking only those objects above an activation value of 0.5.

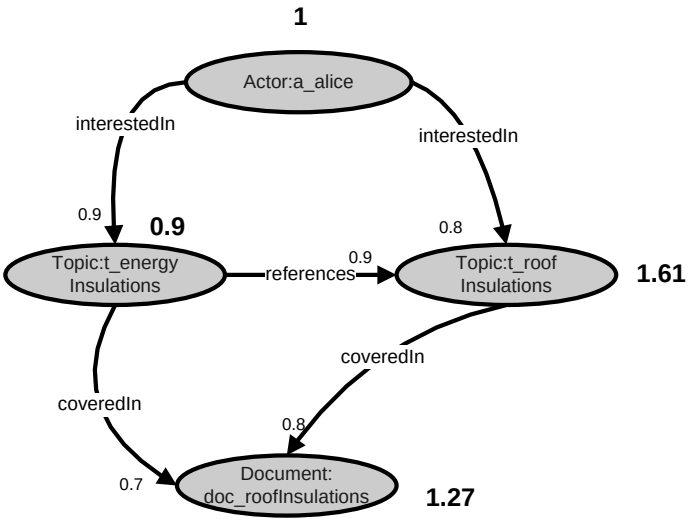

Figure 20 Interpreted state after spreading activation and projection.

Based on the contextualized state, we can formulate an adaptation rule for recommending to *Actor:a_alice* artifacts of interest:

```
//select all documents from the contextualized state with
//an activation > 0.5
documents := getDocumentsWithActivationLargerThan(0.5);
//displaying all documents
IF (notEmpty(documents)) {
  display(documents);
}
```

The function `getDocumentsWithActivationLargerThan` returns from the contextualized state a set of all documents with an activation greater than 0.5. Accordingly, *Actor:a_alice* will get a recommendation to read the document *Document:doc_roofInsulations* which covers some of the topics she is interested in.

After *Actor:a_bob* adds the new document *Document:doc_newRoofInsulations* to the database leading to a new state (see Figure 21), the system has two alternatives for setting the new focus for contextualization: Firstly, it can use the newly added document as the new focus (initial node for the spreading activation algorithm) to select all actors who are highly interested in the topics this artifact covers. Secondly, the newly added document *Document:doc_newRoofInsulations* is recommended to an actor after logging in, thus using the actor itself as the new focus (initial node of spreading activation as shown before).

Using the first alternative, awareness functions are able to directly inform actors interested in *Topic:t_roofInsulations* about the new document *Document:doc_newRoofInsulations* added by *Actor:a_bob*.

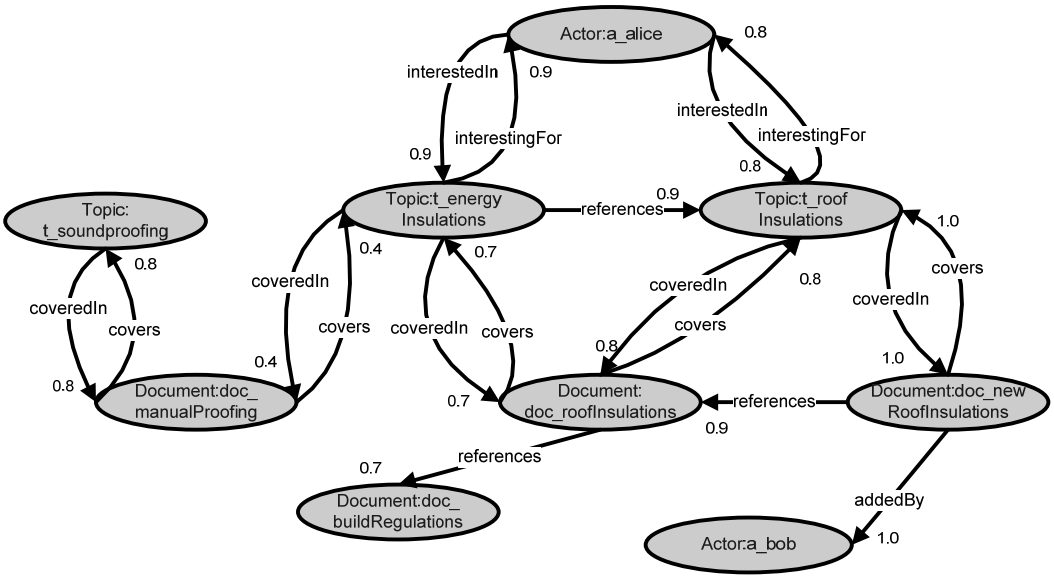

Figure 21 Excerpt of the state after *Actor:a_bob* adds a new document *Document:doc_newRoofInsulations*. Note, that in this figure all relations from the domain model (cf. Figure 17) are included.

The contextualized state after applying the spreading activation is illustrated in Figure 22.

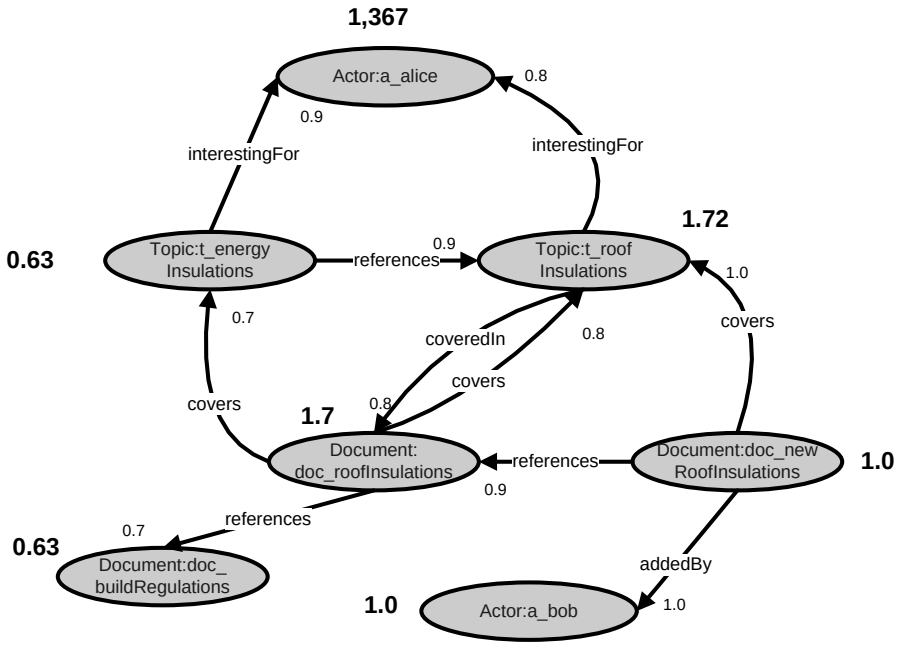

Figure 22 Contextualized state after spreading activation with focus on new document.

Further user-based adaptations can be made using a feedback mechanism. If a user is not satisfied with the recommendations for a specific topic, he can provide feedback which directly decreases the relation strength connecting the artifact with the given topics or to decrease the relation strength between references from artifact to artifact or topic to topic.

6.4. *Co-Dependency*

Co-dependency denotes the situation where several tasks, objects or users are dependent. When considering ill-structured problems, such co-dependencies are often expressed by means of ad-hoc workflows. An ad-hoc workflow helps to structure the implicit processes of information-centered unstructured interaction. Ad-hoc workflows are collaboratively created and explicitly represent the tasks within a team as a shared document. During the co-construction of an ad-hoc workflow, the group members become aware of the required steps for the specific tasks and coordinate their efforts.

During the enactment phase of an ad-hoc workflow, the group members document their progress in the process and thereby increase the awareness of the group's activities. Since group members are allowed to deviate from the ad-hoc workflow they keep the flexibility of information-centered collaboration and can adapt their means to address the ill-structured problem.

Consider the example of a design project where Bob and Alice need to perform several dependent tasks. Each task is defined by its priority, workload, deadline, actor(s), artifact(s), and a task description.

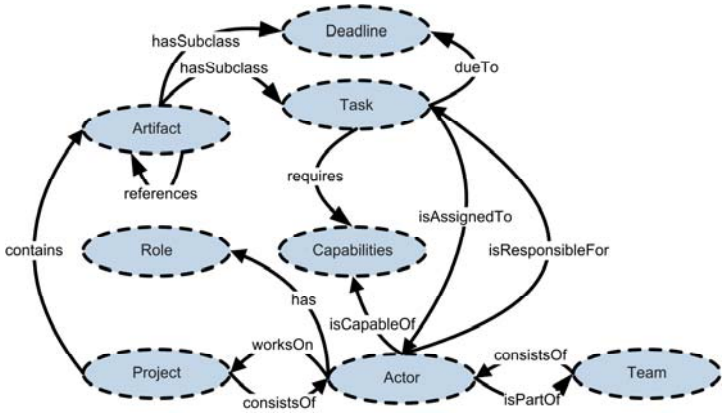

Figure 23 Excerpt from Figure 9 extending the domain model with the classes for “Deadline”, “Task”, “Capabilities”, and “Project”.

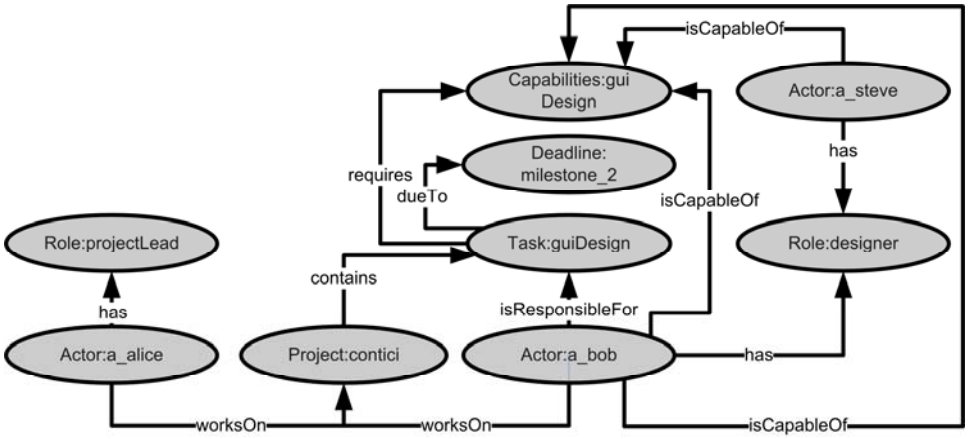

Figure 24 Initial contextualized state

We now discuss some possible adaptations. Let us now imagine the situation where Bob is assigned a high-priority task, which he has not yet started working upon. When the remaining time to the deadline is equal to the workload (i.e. the time needed to perform the task) work on the task needs to commence immediately. In case Bob is not available (e.g. due to sick-leave or vacation), the project would be delayed. We assume that the contextualization block shown below extracted the relevant contextualized state shown in Figure 24 using the focus *Project:contici*. For space reasons we only show context individuals and relations relevant for adaptation in Figure 24. Now, a context

adaptive system could use the context information shown in Figure 24 to find out that such a situation is pending, and could use an adaptation rule helping to resolve the problem. Strategies for resolving the problem include, e.g.:

- *Automatic re-assignment strategy*: Assigning the task to another available worker with similar capabilities.
- *Manual re-assignment strategy*: Presenting this situation (including potential available candidates for replacing Bob on this task) to the project manager for manual resolution.
- *Communication strategy*: Establishing a quick (virtual) meeting among the available project members to discuss and resolve the problem.

The automatic re-assignment strategy could be encoded in an adaptation rule as follows:

```
// Contextualization block, ${focus} = Project:contici
tasks := getTasksInContext(${focus});
members := getTeamMembersInContext(${focus});
capabilities := getCapabilitiesInContext(tasks);
capActors := getCapableActorsInContext(capabilities);
// Adaptation rule
IF (isNotEmpty(tasks)) THEN {
   FOREACH (task : tasks) DO {
      IF (task.priority=high AND
          task.workload = (task.deadline-currentDate()) AND
          isNotAvailable(task.worker))
         // We have a problem since a critical task
         // is not being worked upon in time
      THEN {
         // Assign a matching replacement and
         // inform him/her immediately
         actor := findAvailableReplacement(task,
            capActors);
         assignActorToTask(task, actor);
         notifyActor(actor, task);
      }
   }
}
```

The application of this rule leads to the new adapted state presented in Figure 25. As the figure shows, the relation *isResponsibleFor* between *Actor:a_bob* and *Task:guiDesign* has been removed while other important relations (e.g. *worksOn*) has been created. *Actor:a_steve* is now responsible for this task, works on the *Project:contici*, and has the *Role:designer*.

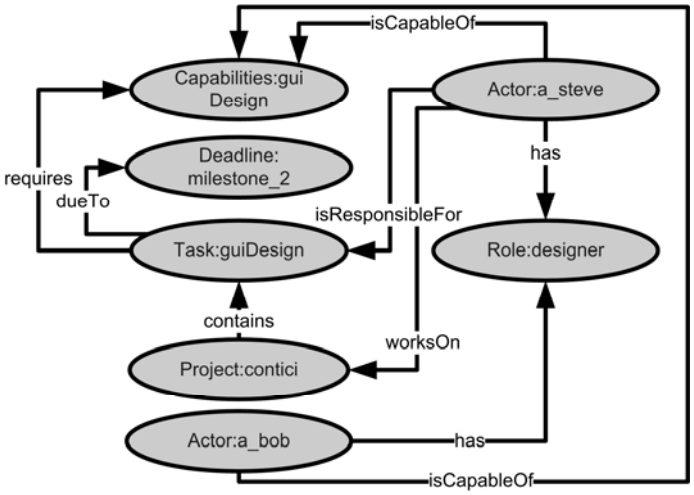

Figure 25 Adapted state (automatic re-assignment strategy)

The supported manual re-assignment strategy could be encoded as follows:

```
// Contextualization block, ${focus} = Project:contici
tasks := getTasksInContext(${focus});
members := getTeamMembersInContext(${focus});
capabilities := getCapabilitiesInContext(tasks);
capActors := getCapableActorsInContext(capabilities);
// Adaptation rule
IF (isNotEmpty(tasks)) THEN {
   FOREACH (task : tasks) DO {
      IF (task.priority=high AND
          task.workload = (task.deadline-currentDate()) AND
          isNotAvailable(task.worker))
         // We have a problem since a critical task
         // is not being worked upon in time.
      THEN {
         // Present problem and auxiliary information
         // to project manager
         manager := task.project.manager;
         actorSet := findAvailableReplacements(task,
            capActors);
         presentInformation(manager, task, actorSet);
      }
   }
}
```

The application of this rule leads to the new state presented in Figure 26. Note, in this case the adaptation rule ends with the display of the case information, possibly in a separate tool displayed to the project manager. In this tool, the project manager may browse the task and available actors in `actorSet`, communicate with them, and decide upon the actor to be used as a replacement. Thus, replacing Bob as worker of the task with, e.g., Steve, is done in this tool and thus outside the adaptation engine. However, once the manager created the new assignment this is reflected in the state and, after respective contextualization, another adaptation rule may be used to inform the new worker immediately about the new and urgent responsibility.

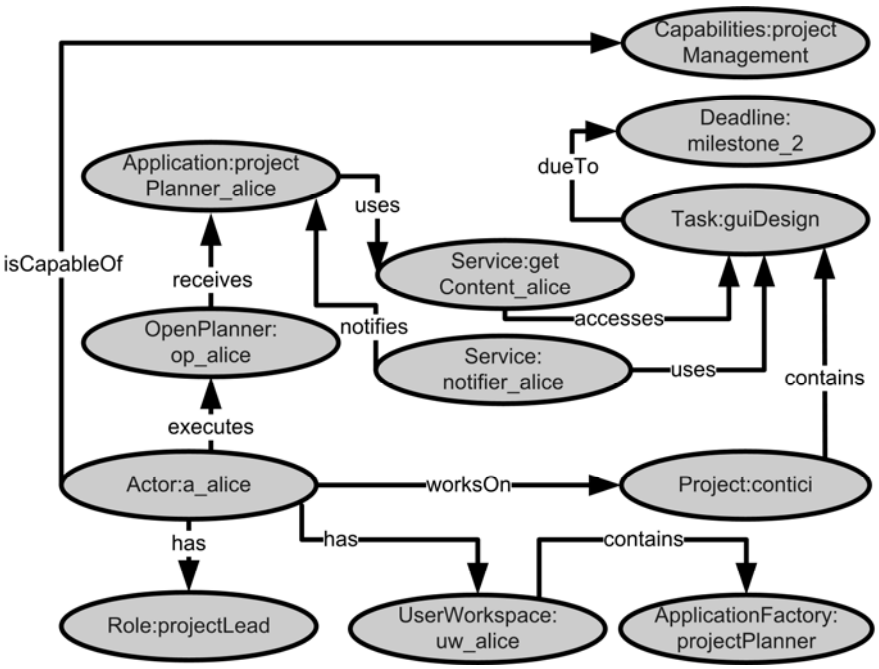

Figure 26 Adapted state (manual re-assignment strategy)

Finally, the communication strategy could be encoded in an adaptation rule by reusing some elements of the prior co-access scenario:

```
// Contextualization block, ${focus} = Project:contici
tasks := getTasksInContext(${focus});
members := getTeamMembersInContext(${focus});
capabilities := getCapabilitiesInContext(tasks);
capActors := getCapableActorsInContext(capabilities);
// Adaptation rule
IF (isNotEmpty(tasks)) THEN {
   FOREACH (task : tasks) DO {
```

```
      IF (task.priority=high AND
          task.workload = (task.deadline-currentDate()) AND
          isNotAvailable(task.worker))
         // We have a problem since a critical
         // task is not being worked upon in time
      THEN {
         // Present problem to available project members
         applications := getApplicationsInContext(members,
                         "Communication");
         IF (isNotEmpty(applications)))
         THEN {
            selApplication := selectOneFrom(applications);
            openForAll(selApplication, members);
            presentInformation(task, members);
      }
   }
}
```

The application of this rule leads to the new state presented in Figure 27. Here, the rule determines the project members and one communication application available to all of them. Then this communication tool is opened for all members, and information about the problem is presented to each member, possibly in a separate tool displayed to each of them.

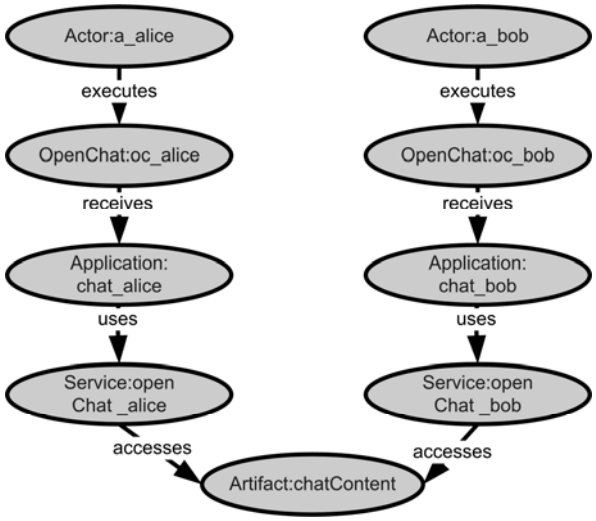

Figure 27 Adapted state (communication strategy); we assume that each member has access to a Chat application as in section 6.2

7. An architecture for context-adaptive collaborative workspaces

In the previous sections we proposed a Generic Context Framework (GCF, cf. Section 4), and a sample domain model for collaborative workspaces (cf. Section 5), and we demonstrated its applicability in typical collaboration situations (cf. Section 6). In this section, we discuss how our approach can be implemented and used to extend collaborative applications with context-based adaptation. Typically, this is done by implementing the core functionality of the framework in an (adaptation) runtime environment and by providing another part for integrating collaborative applications.

The core functionality of GCF consists of the layered context model (cf. Section 4.1), which must be explicitly represented and kept persistent, and the adaptation process (cf. Section 4.2). The adaptation process (cf. Figure 6) consists of three steps (sensing, contextualization, and adaptation), which are implemented in an *Adaptation Server* by using three engines (*Sensing Engine*, *Contextualization Engine*, and *Adaptation Engine*). We use dedicated context services (*GCF Services*) for accessing and manipulating the *Context Model*. The *Context Model* consists of the different *GCF Models* (e.g., the *Domain Model*, the *Sensing Rules*, the current *State (including references to shared models used by the applications)*, the *Contextualization Rules*, the *Contextualized State*, the *Adaptation Rules*, and the *Adapted State*). Figure 26 shows on the right side the architecture of the adaptation runtime environment.

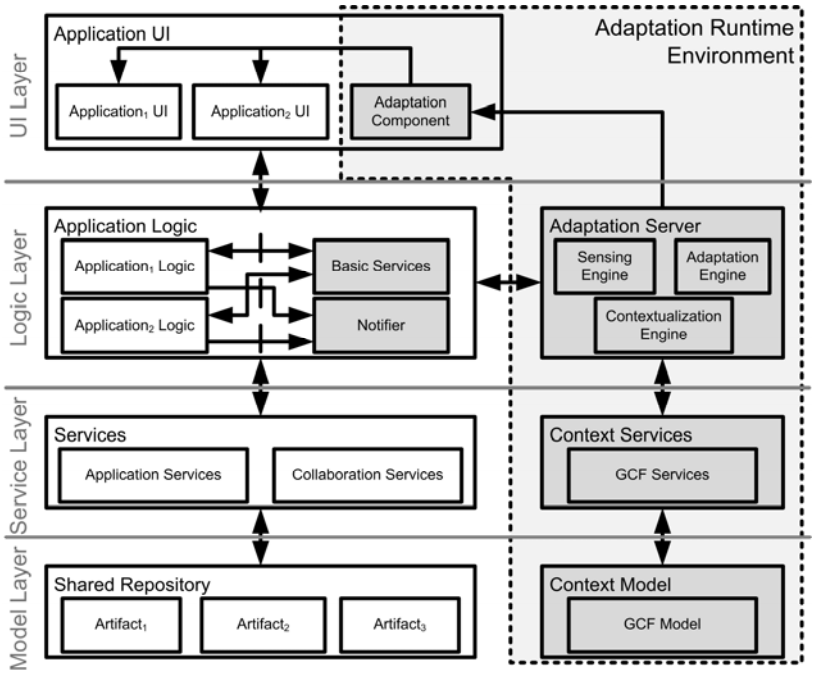

Figure 28 Conceptual Architecture

In order to make an existing collaborative application context-adaptive, we need to connect the application with the runtime environment: Firstly, we need to enable the *Adaptation Engine* to trigger changes of the *Application UI*; this is done via an *Adaptation Component* that calls specific adaptation functions to be provided by the application developer. Secondly, we need to enable the *Sensing Engine* to monitor user interaction and changes of the computing environment; this is done via *Basic Services* that intercept service calls of interfaces that the application developer has registered previously. Thirdly, we need to enable the *Adaptation Engine* to request changes of the *Application Logic*; this is done via *Basic Services*, too. A *Notifier* component is used to inform *Application Logic* and *Adaptation Server* about relevant configuration changes (used for sensing, or used for UI update). Figure 28 shows on the left side the architecture of the application environment and its integration with the *Adaptation Server*.

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).

User interactions usually imply service calls at the corresponding *Application Logic*. In order to monitor these service calls, the developer must register the interfaces an application offers by using *Basic Services*. The calls of these interfaces are intercepted by the *Basic Services* and forwarded to the *Sensing Engine* of the *Adaptation Server* to update the current *State*. To apply adaptation actions that change the service composition of an application, the *Basic Services* have to call the *Application Logic* of the corresponding application via service calls of specific configuration interfaces.

The *Notifier* is used to inform registered components (e.g., components from *Application Logic* or *Adaptation Server*) about specific events (e.g., changes of the configuration, or the status of a user). An application developer can integrate this functionality into his/her application by using the corresponding services. After changing the configuration of an application the *Basic Services* use the *Notifier* to inform registered clients (at least the *Adaptation Component*) that the new configuration is set, and the adaptation of the UI-parts can be performed (e.g., activate a certain view and initialize it).

We implemented the conceptual architecture shown in Figure 26 to support context-based adaptations within a shared work environment based on *Eclipse*[c]. In the current prototype, the *Application UIs* in the *UI Layer* are implemented as plug-ins for *Eclipse.* The *Application Logic* and the *Adaptation Server* in the *Logic Layer* are based on Equinox[d] and realize all components as so-called bundles in *OSGi*[e]. We use *R-OSGi*[f] for

[c] http://www.eclipse.org
[d] http://www.eclipse.org/equinox/
[e] http://www.osgi.org

the communication between the *UI* and the *Logic Layer* as well as for the communication between the *Basic Services* (*Application Logic*) and the *Sensing Engine* of the *Adaptation Server*. The *Services* in the *Service Layer* can be implemented using different techniques (e.g., Web Services, Remote Method Invocation). This layer is used by the *Application Logic* to build the business logic of the application and to access the corresponding *Artifacts* in the *Shared Repository*. The *Context Services* and the *Context Model* are implemented as bundles and reside within the *Adaptation Server*.

8. **Discussion**

Collaborative work is characterized by frequently changing situations and corresponding demands for tool support and interaction behavior provided by the collaboration environment.

Today, collaborating users have to tailor their collaboration environment manually in a consistent and meaningful fashion, leading to a high cognitive overload, ignoring potentials for improvement, and subsequent suboptimal collaboration within the team. Few approaches exist that help to automate or support this adaptation process, e.g., through single-user based adaptation of individual tools. However, issues beyond single-user based adaptation, such as combining individual user models into a group model and adapting collaboration environments for group use remain unanswered. Open issues relate to the definition of context (capturing relevant situations and manipulations in a collaborative environment, formal representation), the support for building context models (formal representation of context and adaptation knowledge), and the support for building context-adaptive systems (separation of concerns, modularity, extensibility, consistency).

In this paper, we presented a generic four layer framework for modeling context in a collaboration environment in a formal way. By separating concerns between knowledge layer capturing knowledge about the collaboration domain as well as the task domain(s), state layer representing a concrete collaboration situation, contextualization layer defining what part of the state is relevant for a given set of focus objects, and adaptation layer representing adaptation rules and the resulting adaptation state, we support modular extension and refinement of context models.

This layered context model is used by a generic adaptation process translating user activity into state, deriving context for a given focus, and executing adaptation rules on this context. The execution of the adaptation rules results in adaptation of the user interface and interaction behavior of the collaboration environment.

Our proposed collaboration domain model (cf. section 5) shows how a collaboration environment can be modeled to allow both, expressing collaboration situations as well as formulating meaningful adaptations for specific situations through adaptation rules. In section 6 we described how our approach can be used to support context-based adaptation

[f] http://r-osgi.sourceforge.net/

in four typical collaboration situations: co-location, co-access, co-recommendation, and co-dependency.

Finally, section 7 described a conceptual system architecture for implementing context- adaptive collaboration environments and reported about our prototypical implementation.

Our approach exceeds current approaches by making first progress towards how individual user models can be flexibly combined into a group model through representing the needed aspects on the state layer and defining group context through contextualization rules in the contextualization layer. Open issues still remain such as solving conflicting adaptation requirements, different user preferences, or calculating group interest profiles from individual profiles. Furthermore, the proposed model of a collaboration environment and its representation in the collaboration domain model facilitates the definition of adaptation rules that can be applied in specific collaboration situations (e.g. co-access) and that define meaningful adaptation behavior across tools and users of the environment (e.g. opening communication tools, establishing sessions, and showing helpful awareness information). Both, the four layer context modeling framework and the collaboration domain model were validated by applying them in four typical collaboration situations.

This work provides the basis for further research into context-based adaptive systems, including:

- comparison of effects of different adaptation rules in the same collaboration situation (leading to best practice know how),
- comparing applicability and performance of different contextualization and adaptation approaches (which can be encoded in the respective adaptation server),
- comparing the effect of using different user interfaces and interaction features within the collaboration environment and its tools, and
- studying learning curves and long-term performance development within collaborating teams with and without context-based adaptations.

Acknowledgment

This work is supported by the German Research Foundation (DFG) within the cluster project “Context Adaptive Interaction in Cooperative Knowledge Processes” (CONTici).

References

1. G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper and M. Pikerton, Cyberguide: a mobile context-aware tour guide, *Wireless Networks* Nr. 5 Vol. 3, 1997, pp. 421-433.
2. H. J. Ahn, H. J. Lee, K. Cho and S. J. Park, Utilizing knowledge context in virtual collaborative work, *Decision Support Systems*, Nr. 4, Vol. 39, 2005, pp. 563-582.
3. J. R. Anderson, A spreading activation theory of memory. *Journal of Verbal Learning and Verbal Behavior*, *22* (1983), pp. 261–295.
4. J. L. Austin, *How to Do Things with Words* (Oxford: O.U.P., 1962).
5. U. M. Borghoff, and J. H. Schlichter, *Computer-Supported Cooperative Work*, Springer-Verlag Berlin Heidelberg New York, 2000.
6. P. Brèzillon, Using Context for Supporting Users Efficiently, in *HICSS’03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS’03)* – Track 5,

2003, pp.127.3.
7. H. Bunt, Context and Dialogue Control, *THINK Quarterly*, Vol. 3, 1994.
8. L. Buriano, M. Marchetti, F. Carmagnola, F. Cena, C. Gena and I. Torre, The Role of Ontologies in Context-Aware Recommender Systems, in *Proceedings of the 7th International Conference on Mobile data Management (MDM'06)*, 2006.
9. R. Carnap, *Meaning and Necessity: A Study in Semantics and Modal Logic*, 2nd Ed., 1988.
10. H. Chen, T. Finin and A. Joshi, Using OWL in a Pervasive Computing Broker, *Workshop on Ontologies in Agent Systems*, 2003.
11. H. Chen, T. Finin and A. Joshi, Semantic Web in the Context Broker Architecture, in *Proceedings of the Second IEEE Internation Conference on Pervasive Computing and Communications (PerCom'04)*, 2004.
12. G. Chen and D. Kotz, A survey of context-aware mobile computing research, *Dartmouth College Technical Report*, 2000.
13. P. R. Cohen and R. Kjeldsen, Information retrieval by constrained spreading activation in semantic networks. *Information Processing and Management*, *23*(2) (1987), pp. 255–268.
14. A. M. Collins and E. F. Loftus, A Spreading Activation Theory of Semantic Processing. *Psychological Review*, *82*(6) (1975), pp. 407–428.
15. M. Constantino-Gonzalez and D. D. Suthers, Automated Coaching of Collaboration Based on Workspace Analysis: Evaluation and Implications for Future Learning Environments, *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) – Track 1* (2003).
16. F. Crestani, Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, *11*(6) (1997), pp. 453–482.
17. G. M. Davies and D. M. Thomson, *Memory in Context; Context in Memory* (John Wiley & Sons Inc., 1988), pp. 1-10.
18. A. K. Dey, G. D. Abowd, P. J. Brown, N. D. Davies, M. Smith and Pete Steggles, Towards a Better Understanding of Context and Context-Awareness, in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, 1999, pp. 304-307.
19. A. K. Dey, Understanding and using Context, *Personal Ubiquitous Computing* Nr. 1 Vol. 5 (2001), pp. 4-7.
20. A. K. Dey, R. Hamid, C. Beckmann, I. Li and D. Hsu, a CAPella: Programming by Demonstration of Context-Aware Applications, in *Proceedings of the SIGCHI conference on human factors in Computer Systems (CHI'04)*, 2004, pp. 33-40
21. C. Dorn and S. Dustdar, Sharing hierarchical context for mobile web services, *Distributed parallel Databases* Nr. 1 Vol. 21 (2007), pp. 85-111.
22. C. Dorn, H. Truong and S. Dustdar, Measuring and Analyzing Emerging Properties for Autonomic Collaboration Service Adaptation, in *Proceedings of the 5th international conference on Automatic and Trusted Computing (ATC'08)*, 2008, pp. 162-176.
23. P. Dourish, What we talk about when we talk about context, *Personal Ubiquitous Computing* Nr. 1 Vol 8 (2004), pp. 19-30.
24. W. K. Edwards, Putting Computing in Context: An Infrastructure to support extensible context-enhance collaborative Applications, *ACM Trans. Computer-Human Interactions* Nr. 4 Vol. 12 (2005), pp. 446-474.
25. L. Fuchs, AREA: a cross-application notification service for groupware, in *Proceedings of the sixth conference on Computer Supported Cooperative Work (ECSCW'99)*, 1999, pp. 61-80.
26. T. Gross and W. Prinz, Modelling Shared Contexts in Cooperative Environments: Concept, Implementation, and Evaluation, *Computer Supporter Cooperative Work*, Vol. 13, 2004, pp. 3-4.
27. J. M. Haake, T. Schuemmer, 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 (HICCS-37)*, 2004.
28. J. M. Haake, A. Haake, T. Schuemmer, M. Bourimi and B. Landgraf, End-User Controlled Group Formation and Access Rights Management in a Shared Workspace System, in *Proceed-*

ings of the 2004 ACM conference on Computer supported cooperative work (CSCW'04), 2005, pp. 554-563.
29. A. Huegli and P. Luebcke, *rororo Philosophielexikon*, 2001.
30. T. Hussein, D. Westheide and J. Ziegler, Context-adaptation based on Ontologies and Spreading Activation, *LWA 2007 : Lernen – Wissen – Adaption*, ed. A. Hinneburg, 2007, pp. 361-366.
31. J. W. Kaltz and J. Ziegler, A conceptual model for context-aware Web Engineering, In *Proceedings Workshop on Modelling and Retrieval of Context*, Vol. 114, 2004.
32. J. W. Kaltz, J. Ziegler and S. Lohmann, *RIA – Revue d'Intelligence Artificielle, Special Issue on Applying Context-Management*, 19(3), S. 439-458, 2005.
33. A. Kobsa, Generic User Modeling Systems, in *Methods and Strategies of Web Personalization*, eds. P. Brusilovsky, A. Kobsa and W. Neidl (Springer Verlag, Berlin, Heidelberg, New York, 2007), pp. 136-154.
34. G. E. Krasner and S. T. Pope, A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, *Journal of Object-Oriented Programming* Nr. 3 Vol. 1 (1988), pp. 26-49.
35. B. MacIntyre, E. D. Mynatt, S. Voida, K. M. Hansen, J. Tullio and G. M. Corso, Support for Multitasking and background awareness using interactive peripheral displays, in *Proceedings of the 14th annual ACM symposium on user interface software and technology (UIST'01)*, 2001, pp. 41-50.
36. B. Malinowski, The Problem of Meaning in primitive Languages, *The Meaning of Meaning* (1923), pp. 146-152.
37. M. A. Martinez-Carreras, A. Ruiz-Martinez, A. F. Gomez-Skarmeta and W. Prinz, Designing a Generic Collaborative Working Environment, In *Proceedings of IEEE International Conference on Web Services*, 2007, pp. 1080-1087.
38. D. D. Mittleman, R. O. Briggs, J. Murphy and A. Davis, Toward a Taxonomy of Groupware Technologies, in *Proceedings of the 14th Collaboration Researchers' International Workshop on Groupware (CRIWG 2008)*, 2008, pp. 307-321.
39. W. Prinz and B. Zaman, Proactive support for the organization of shared workspaces using activity patterns and content analysis, *in Proceedings of the 2005 international ACM GIGGROUP conference on supporting group work (GROUP'05)*, 2005, pp. 246-255.
40. W. Prinz, H. Loh, M. Pallot, H. Schaffers, A. Skarmeta and S. Decker, ECOSPACE – Towards an Integrated Collaboration Space for eProfessionals, in *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2006, pp. 39-45.
41. M. Rittenbruch, Atmosphere: towards context-selective awareness mechanisms, in *Proceedings of the 8th International Conference on Human-Computer Interaction*, 1999, pp. 328-332.
42. J. Roth, *Mobile Computing* (dpunkt.verlag, 2nd. Edition, 2005).
43. G. Salton and C. Buckley, On the use of spreading activation methods in automatic information. In Yves Chiaramella (Ed.), *Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 147–160) 1988. Grenoble, France: ACM.
44. D. Schall, H. Truong and S. Dustdar, Unifying Human and Software Services in Web-Scale Collaborations, *IEEE Internet Computing* Nr. 3 Vol. 12 (2008), pp. 62-86.
45. B. Schilit, N. Adams and R. Want, Context-Aware Computing Applications, *(IEEE) Workshop on Mobile Computing Systems and Applications*, 1994.
46. T. Schuemmer and S. Lukosch, *Patterns for Computer-Mediated Interaction* (John Wiley & Sons, Ltd., 2007).
47. T. Strang and C. Linnhoff-Popien, A Context Modeling Survey, *Workshop on Advanced Context Modelling, Reasoning and Management*, The Sixth International Conference on Ubiquitous Computing, 2004.
48. S. Voida, E. D. Mynatt, B. MacIntyre and G. M. Corso, Integrating Virtual and Physical Context to Support Knowledge Workers, *IEEE Pervasive Computing*, Nr. 3, Vol. 1, 2002, pp. 73-79.

49. M. Vonrueden and W. Prinz, Distributed Document Contexts in Cooperation Systems, in *Modeling and Using Context, 6th International and Interdisciplinary Conference (CONTEXT 2007)*, 2007, pp. 507-516.
50. X. H. Wang, D. Q. Zhang, T. Gu and H. K. Pung, Ontology Based Context Modeling and reasoning using OWL, in *Proceedings of the Second IEEE Annual conference on Pervasice Computing and Communications Workshops*, 2004.
51. T. Winograd, Architectures for Context, *Human-Computer Interaction*, Vol. 16 (1991), pp. 401-419
52. V. Wulf and B. Golombek, Exploration Environments – Concepts and Empirical Evaluation, in *Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work (GROUP'01)*, 2001, pp. 107-116
53. T. Ziemke, Embodiment of Context, in *Proceedings of ECCS*, 1997.
54. A. Zimmermann, A. Lorenz and R. Oppermann, An Operational Definition of Context, *Modeling and Using Context* (2007), pp. 558-571.