A Context-based Web Service Approach to Communities of Practice

Renato de Freitas Bulcão Neto[1]
Carlos Henrique Odenique Jardim[1]
José Antonio Camacho Guerrero[1]
Daniel Corrêa Lobato[1,2]
Maria da Graça Pimentel[1]

Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo
Av. Trabalhador São-Carlense, 400 – Caixa Postal 668
13560-970, São Carlos, SP

[2]Faculdades COC
R. Abraão Issa Halack, 980
14096-000, Ribeirão Preto, SP

{rbulcao,cjardim,jcamacho,dclobato,mgp}@icmc.usp.br

Abstract. *Non-structured groups of people with common interests are called Communities of Practice (CoP), where members learn by means of their social participation in tasks related to the community. Applications that support CoP share many common requirements with Computer-Supported Collaborative Work (CSCW) applications and context-aware applications. We present an infrastructure that can be used to manage context information in CoP/CSCW applications and illustrate its use by a web-based recommendation system.*

Resumo. *Grupos não estruturados de pessoas com interesses em comum são chamados de* Comunidades de Prática *(CoP), onde um membro aprender através de sua participação social nas atividades relativas à comunidade. Aplicações que suportam CoP apresentam muitos requisitos em comum com aplicações de trabalho colaborativo suportado por computador (CSCW) e aplicações cientes de contexto. Apresentamos uma infraestrutura que pode ser utilizada para gerenciar informações de contexto em aplicações CoP/CSCW e ilustramos seu uso através de um sistema de recomendação baseado em* Web.

Keywords: Communities of Practice, CSCW, Context-awareness, Web Services.

1. Introduction

The need for knowledge demanded from individuals as well as from organizations not always arise from structured processes of knowledge management. Such need may result from groups of people with common interests and different expertise who can collaborate in order to solve complicated problems or situations. These non-structured groups of people with common interests are known [Lave and Wenger, 1991] as Communities of Practice (CoP).

Members of CoP learn by means of their social participation in tasks related to the community. As time goes by, the role of members is shifted from a peripheral participation to a more active perspective in the community, including the transmission of knowledge to new members. This kind of knowledge transformation resembles the Knowledge Spiral [Nonaka and Takeuchi, 1995], where new knowledge is created from existing one. Instead of enforcing physical presence, visible limits or well-defined groups, CoP enforce that their members spontaneously participate in tasks with common goals, sharing their needs and problems.

In organizations the objectives of communities and projects are orthogonal. The participation in a project is defined by the organization based on technical criteria. On the other hand, CoP reflect the interest and expertise of their members who are free to join or to leave a community. Furthermore, the role of a member can be explicitly defined by the community itself, or partially deduced by the system based on members' behavior — whereas organizations define the role of members a priori.

The characteristic of informality in the creation of CoP is notably identified when the workplace is carefully observed. Most of the information is usually captured in a non-structured way from informal activities such as coffee breaks conversations.

Transferring the concept of community to a *virtual community*, such as a computer programm, can be a hard task involving some major problems, including but not exclusively conceptual mapping. Congar et al. ([Congar et al., 1999]) argues that the use of MUD (*Multi-user dungeon*) can be an effective way of mapping the community to a virtual environment supported by computers: the users can really "see" and "talk" to other users, as if they were near they.

Even if the graphical abstraction of community is not used, the sense of community can be preserved. The Usenet newsgroups seems to be a community ([Roberts, 1998]). Same aspects that corroborates this assertion is the presence of shared terms of good conduct (the so-called *netiquete*), shared humor, the definition of a member's role based on his behavior (the newsgroups' moderators), and organized real-world meetings when applicable.

By using a computer supported environment to the CoP, the members of CoP does not need to be physically close to each other; thus, one member can interact with members that are hundreds or thousands of kilometers away, even in other state or country. This help on spreading knowledge beyond a region or a group of people geographically close to each other.

The literature reports on efforts targeted at supporting such communities that exploit the Web platform (e.g. [Millen and Fontaine, 2003] and [Mack et al., 2001]). In some cases, profile and other contextual information may be used to help users to identify information of common interest such as in MILK (*Multimedia Interaction for Learning and Knowing*) [Agostini et al., 2003]. Considering this perspective, applications supporting CoP share many requirements with typical context-aware applications that rely on the ability of a computational entity to adapt its behavior based on context information sensed both from the physical and computational environment [Dey, 2001].

However, support to CoP applications also share many common requirements with Computer-Supported Collaborative Work (CSCW) applications. For instance, CSCW applications depend on the availability of information with respect to individuals and the groups that they belong to (group awareness) in order to provide the most basic capabilities for social awareness, which "includes information about the presence and the activities of people in a shared environment" [Prinz, 1999].

The need for infrastructures to support building CSCW applications has been long discussed. In some cases, the components are associated with architectural models so as to facilitate the design as well as the evolution of applications such as the case of Dragonfly [Anderson et al., 2000] and Clover [Laurillau and Nigay, 2002]. Efforts have also been geared towards providing CSCW applications with generic infrastructures handling event notification [Prinz, 1999] [Fitzpatrick et al., 1999] [Patterson et al., 1996], and specifically sharing context information [Rittenbruch, 1999] [Fuchs, 1999]. The main focus is on supporting building integrated CSCW applications such as the use of context information by shared workspaces [Gross and Prinz, 2003].

In previous work [Arruda Jr et al., 2003] we have shown how context-aware CSCW applications can take advantage of the benefits provided by Web Services [W3C, 2002], which have become a key success factor as infrastructure for several development and integration projects [Arsanjani et al., 2003]. The essence of Web Services is the use of the Internet infrastructure to bridge a myriad of Internet systems transparently and independently of differences in network technologies, devices, operating systems and programming languages. Web Services are largely based on HTTP as the application-level protocol, and open XML-based specifications, such as XSD (*XML Schema Definition Language*) [Fallside, 2001], WSDL (*Web Service Description Language*) [Chinnici et al., 2004] and SOAP (*Simple Object Access Protocol*) [Mitra, 2003].

In this paper we have extended our previous modeling and implementation relative to the classic *who* and *where* context dimensions so as to support requirements demanded by CSCW applications in general and CoP applications in particular — from now on called CoP/CSCW applications.

We discuss related work relative to support CoP, CSCW and context-aware applications in Section 2. We introduce our extended version of the Context Kernel in Section

3 and illustrate how its use by a web-based recommendation system can support typical CoP in Section 4. We present our conclusions and future work in Section 5.

2. Related Work

The literature reports that in information technology (IT) organizations most of the members of CoP use websites and email as a tool for acquisition or transformation of knowledge [Millen and Fontaine, 2003]. Collaborators of an IT organization have reported positive results by using taxonomies in their portal to organize the knowledge about their clients and to allow the exchange of knowledge for problem solving [Mack et al., 2001].

The *Multimedia Interaction for Learning and Knowing* (MILK) aims at allowing CoP to manage the knowledge produced by a community [Agostini et al., 2003]. The knowledge organization in MILK uses a profiling mechanism that allows the association of knowledge descriptors to objects of different sources (called documents). That mechanism is based on three types of metadata: generic (e.g. author's name, creation date, type), ontology-based content, and qualifiers (e.g. recommendations of users, relevance of documents). When a user publishes a document in MILK, it appears within a context based on the profiling generated using metadata. An interface called *View With Context* (VWC) automatically creates relationships with other documents in the same context, which facilitates information discovery by users.

CoP applications share many requirements with typical context-aware applications. However, making applications context-aware is one of the challenges pointed out in the ubiquitous computing literature [Abowd et al., 2002], which reports results on mobile and context-aware toolkits, frameworks and service infrastructures.

Context Toolkit [Dey, 2001] offers a framework with generic services and abstractions in order to overcome the lack of support for a standard development of sensor-based context-aware applications. The framework has been used to build CSCW applications including the *Conference Assistant* [Dey et al., 1999] that supports groups of users participating in a large conference. The handling of events and sensors has been implemented in the NESSIE generic infrastructure for contextual event notifications [Prinz, 1999], while the modeling of context information so as to map incoming events to a context of origin is the core of other proposals [Gross and Prinz, 2003] [Fuchs, 1999] [Rittenbruch, 1999].

The *Aura* project [Garlan et al., 2002] provides a framework for user mobility in ubiquitous computing environments. Some CSCW applications that exploit Aura include the *Portable Help Desk* (PHD) [Salber et al., 2001], which allows a mobile student on a campus to locate his colleagues and find useful resources (e.g. printers and restaurants).

CoP applications also share many common requirements with CSCW applications. Several efforts have been directed towards providing CSCW applications with generic infrastructures handling event notification to provide task awareness such as NESSIE [Prinz, 1999], NSTP (*Notification Service Transfer Protocol*) [Patterson et al., 1996] and *Elvin* [Fitzpatrick et al., 1999]. Regarding the modeling of context information, the literature needs models as elaborate specifications that must be provided by applications — or the user — so that the associated rules can be computed by the corresponding infrastructures. Such effort corresponds to the tradeoff of having a generic service manipulating the context information. We report our efforts towards supporting requirements demanded by CSCW applications in general and by CoP applications in particular.

3. The Context Kernel Web Service

The essence of Web Services is the use of the Internet infrastructure to allow applications to communicate seamlessly and independently of heterogeneous hardware and software. Web Services are accessed using standard Internet protocols as HTTP operating on top of TCP, and can be defined by self-describing messages referencing information to understand the message. Open XML-based specifications are the building blocks of the basic Web Services architecture [Burner, 2003].

The XSD language [Fallside, 2001] offers a collection of data types so as to describe XML attributes and elements aiming syntactic interoperability across the Internet. For the exchange of messages Web Services may use the SOAP protocol [Mitra, 2003], which defines the XML-based syntax, the semantics and the order of messages exchanged between peers — applications or services. The WSDL specification [Chinnici et al., 2004] defines the collection of messages which a service accepts and produces. Moreover, WSDL describes the mapping between abstract messages to particular objects or methods to be used by applications requesting the services.

3.1. Representation schema of context information

The Context Kernel is a Web Service that allows applications to handle context information based on the classic dimensions [Abowd et al., 2002]: *who*, *where*, *when*, *what*, *why* and *how*. It classifies those dimensions as follows: (i) *primitive dimensions*, i.e., those handled independently of other dimensions; (ii) *derivative dimensions*, i.e., those obtained by relating other dimensions — primitive or derivative [Arruda Jr et al., 2003].

The Context Kernel stores *primitive dimensions* by means of a premise defined by a tuple containing *type, value* and an optional *qualifier*. The following XML excerpt represents an instance of the primitive dimension *who*:

```
<premise dimension="who" type="login" value="mgp" qualifier=""/>
```

A *derivative dimension* is defined by means of a *rule* that contains at least one *premise* and one *inference*: a *premise* is defined as above and *rules* are grouped in *schemas*. A set of related schemas is then contained in an element *context*.

Following that classification, any dimension can be primitive or derivative, being strictly dependent on the application requirements. Therefore, the applications themselves are responsible for the specification of which kind of data and rules are particularly relevant to them, as it is the case with the other infrastructures reported in the literature.

Example 1 illustrates the vocabulary defined by Context Kernel: a sensor-based application requests "the three most recent (*last value="3"*) activities (*dimension="what"*) which a *person* called *Daniel Lobato* carried out at the *lab4* room on *June 4th, 2003*".

```
<!--            Example 1            -->
<last value="3"/>
<context>
  <premiseS>
    <boolean type="AND">
      <premise dimension="who" type="name" value="Daniel Lobato"/>
      <premise dimension="where" type="room" value="lab4"/>
      <premise dimension="when" type="date" value="2003-06-04"/>
    </boolean>
  </premiseS>
  <inferenceS>
      <inference dimension="what" type="activity"/>
  </inferenceS>
</context>
```

3.2. The Context Kernel architecture

As a typical Web Service, the Context Kernel architecture can be thought as having five distinct layers. In the top, the *application* layer (i) corresponds to applications making use of the Context Kernel service by invoking SOAP messages. The next two layers refer to the *contract* layer (ii) using WSDL for the service description, and the *protocol* layer (iii) using SOAP on top of HTTP, respectively.

The *functional service* layer (iv) describes the service interface, i.e., the main functions provided by the Context Kernel service, such as registering of applications and storing and retrieving context information. The lower *implementation* layer (v) refers to the logic implementation of the Context Kernel, which adopts a document-oriented API implementation abstracting the system architectures and creating a loosely-coupled connectedness that withstands changes to its underlying implementation.

3.3. Using the Context Kernel

The Context Kernel publishes its WSDL document definition that contains the description of the service so that it is available to interested applications. For the case that a specific group of applications wants to make exclusive use of the service — for security or privacy reasons, for instance, or just for relevance relative to the applications domain — this publication can be carried out in restrict forms.

Once a context-aware application designer retrieves the WSDL document, she obtains the service contract and the network endpoints that honor it. In the Context Kernel environment endpoints are Java servlets used to extend the capabilities of servers in a *request-response* programming model.

Once knowing the WSDL specification, the designer builds an application that uses the Context Kernel Web Service as follows: (i) based on the service contract, applications send HTTP messages using the SOAP packaging protocol for requesting the available services; (ii) the Context Kernel stores/retrieves context information provided by applications on/from a repository. The result of the processing of context information is returned to the requesting application via SOAP messages.

As far as the exchange of context information among loosely-coupled applications is concerned, an application: (i) obtains the public identifiers of other applications registered with the Context Kernel; (ii) then retrieves the set of rules stored by those applications using the GetRules service (detailed below). That mechanism allows application designers to be able to write applications that share context information.

3.4. The Context Kernel API

The software platform used in the implementation of the Context Kernel API includes the Java programming language, tools to handle XML-based specifications — XML, XSD, WSDL and SOAP — and the PostgreSQL database.

The Context Kernel API offers five categories of services: *registry, event notification, status, storage* and *retrieval*. Given the Web Services approach, all those services are invoked by applications using the SOAP protocol encapsulating XML-based messages.

Due to space limitations we show practical examples of messages exchanged between applications and the Context Kernel in Section 4.

3.4.1. Registering with the Context Kernel

In order to use the Context Kernel, an application initially needs to register its own information to made it available to other applications. An application invokes the InfoApp service to *register* its metadata (e.g. *name, description* and *developers*) and *receives back* one public and one private identifier. The private identifier is used to store context information; the public identifier is used by third-party applications to access the context information in a read-only basis. Once an application A holds the public identifier of an application B, the application A can use the other services of the Context Kernel API to query the Context Kernel to obtain information stored by application B.

3.4.2. The Context Daemon Notification Service

An important limitation resulting from using the Web Services approach is the lack of support to *notification services* — a most important requirement to CSCW toolkits and generic infrastructures. To overcome this problem, the Context Kernel includes a notification service called *Context Daemon*.

Any application demanding event notification must specify, during registration with the Context Kernel via InfoApp, the callback address by means of which the Context Daemon will be able to send results when they become available.

As a result, the Context Kernel will try to validate all rules stored by that application on the repository (via PutRules service described later). Thus, when any information is stored by the application on the repository, the Context Kernel will check if any of its rules have been met. Although this is quite a consuming approach, it may be necessary for some applications. An alternative approach is being built so that an application will be able to specify which rules should be monitored and when they should be monitored.

3.4.3. Checking the Applications Registered with the Context Kernel

An application can retrieve *status* information stored by another application. This can be achieved by invoking the StatusApps service which returns the public identifiers and the metadata of *all* applications registered with the server. Until the time of this writing, the Context Kernel does not implement a discovery service yet.

Once an application obtains the public identifiers of other applications registered with the Context Kernel, that application can retrieve the set of rules stored by them by means of the GetRules service (described below) — which is the mechanism that allows applications to share and exchange information.

3.4.4. Storing Information on the Context Kernel

The Context Kernel provides services which allow applications to *store* and *retrieve* context information relative to *rules* of context with *premises* and *inferences,* as shown in Example 1. It is worth noting that Context Kernel relies on applications regarding the validity of the data and rules being stored. Moreover, the relevance of each piece of context information or rule of context is prerogative of the applications themselves.

The PutData service allows applications to *store* context information relative to *primitive dimensions* only. The PutRules service allows the storage of context information relative to *rules* of context, thus both *primitive* and *derivative dimensions*. Applications may associate an expiration date for their rules. When its validation date expires, the rule will continue stored on the server, but with solely historical purpose.

3.4.5. Retrieving Information from the Context Kernel

Context Kernel makes available the services GetRules, GetAny and GetInverse for querying context information. The GetRules service allows applications to retrieve the *rules* associated to context information stored by other applications based on the public identifier of that application. This allows that one application be able to obtain the formal specification of the information stored by other applications, which is necessary to applications designers to understand the semantic of the rules and to specify appropriate queries. As a result, applications designers are able to write applications that share context information.

The GetAny service allows applications to retrieve context information based on the value of the *premises* or the *inferences* associated to *primitive* and *derivative dimensions*. Moreover, it is possible to specify responses based on the dimensions themselves. As illustrated in Example 1, applications may also specify the maximum number of answers (*last value*=“ ”) or even combine premises using boolean operators (*boolean type* =“ ”).

The GetInverse service allows applications to retrieve context information using the *inference* to obtain the corresponding *premises*. Moreover, it is also possible to specify responses based on the dimensions themselves. GetInverse follows the semantics of a backward reasoning, while GetAny can be thought as a forward reasoning.

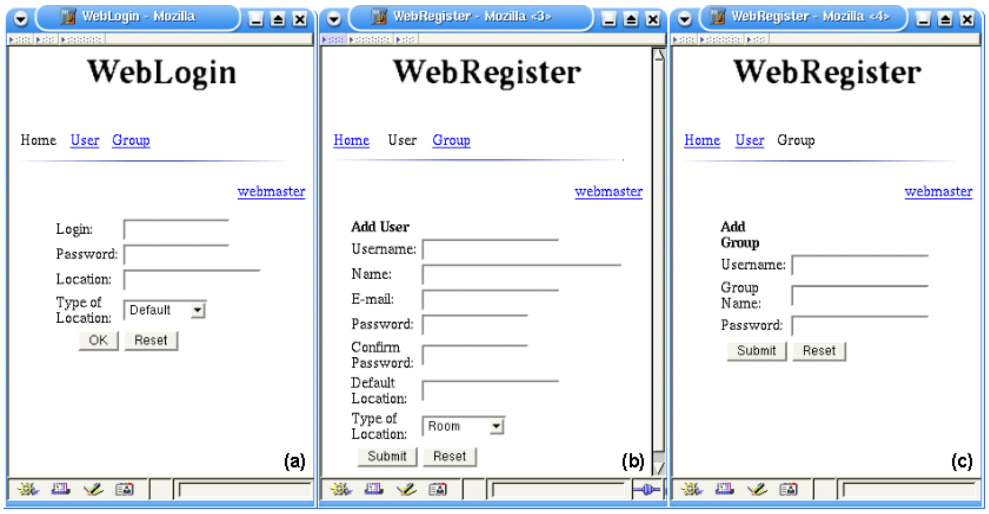


Figure 1: (a) User information will be verified in the Context Kernel; (b) a new user provides the Context Kernel with personal and location information (e.g. her default location); (c) the interface for registering groups.

4. Context Kernel in use by CSCW applications

This section illustrates how applications that support Communities of Practice (CoP) in particular, and CSCW applications in general, can take advantage of the benefits leveraged by Context Kernel. We have built an infrastructure that supports CoP/CSCW by means of applications that exploit users, groups and location information stored in the Context Kernel. We briefly present each application and illustrate its use of the Context Kernel.

4.1. The WebLogin application: informing *Who+Where+When+What*

WebLogin (Figure 1(a)) is an application used as entry point for other applications in our CoP/CSCW environment. The first thing for a user to do is to log in; the typical scenario is that the user will log in after having started some other application, since the *WebLogin* is a component that can be activated by other applications.

Before using the Context Kernel (from now on called CK), *WebLogin* registers itself by calling the InfoApp service. CK generates a public and a private identifier for the application and return those as a result of the call for InfoApp. *WebLogin* calls the StatusApps service to get the public identifier of *WebRegister* (Figure 1(b)). In fact, all applications intending to query user and group information must obtain the public identifier of the *WebRegister* application (as detailed later).

When a user logs in, *WebLogin* verifies if the current user is already registered with the CK database. Group information is stored as *what* inferences while user information is handled as *who* premises. *WebLogin* invokes the GetAny service that searches for some rule which includes a *what* inference which type is group and the premises include the current username and the corresponding password. The result of the query in Example 2 is if the user exists or not. Example 2 also illustrates that users have a group with their own name which allows them to register with CK before registering with any group.

```
<!--           Example 2           -->
<context>
  <premiseS>
    <boolean type="AND">
      <premise dimension="who" type="login" value="cjardim"/>
      <premise dimension="who" type="password" value="********"/>
    </boolean>
  </premiseS>
  <inferenceS>
    <inference dimension="what" type="group"/>
  </inferenceS>
</context>
```

In many CoP/CSCW applications users should inform their current location — either explicitly (Figure 1(a)) or implicitly such as in environments instrumented with sensors. Location information is stored as a *where* premise associated with the current user. *WebLogin* then invokes the PutRules service to store the location information and the login time. Example 3 describes the message required to store the location information of the user "cjardim" who logged from "room" "3-009".

```
<!--           Example 3           -->
<context>
  <premiseS>
    <premise dimension="who" type="login" value="cjardim"/>
    <premise dimension="when" type="date" value="2004-02-27T15:6:21"
             qualifier="datetime"/>
  </premiseS>
  <inferenceS>
    <inference dimension="where" type="room" value="3-009"/>
  </inferenceS>
</context>
```

4.2. The WebRegister application: managing with *Who+Where+When+What*

WebRegister is an application designed for the management of users and groups (Figure 1(b)) and is started by *WebLogin* in the case of new users. *WebRegister* provides its private identifier along with the information regarding to the new user as parameters for invoking the PutRules service. The user should also inform a default location defined as "the most probable location where the user can be found"; this can be used by CoP/CSCW services to provide personalization. The premises relative to the new user are then registered with the CK database as shown in Example 4.

```
<!--           Example 4           -->
<context>
  <premiseS>
    <premise dimension="who" type="login" value="cjardim"/>
    <premise dimension="who" type="name" value="Carlos Jardim"/>
    <premise dimension="who" type="password" value="********"/>
    <premise dimension="who" type="email" value="cjardim@icmc.usp.br"/>
    <premise dimension="where" type="room" value="1-008"/>
  </premiseS>
  <inferenceS>
    <inference dimension="what" type="group" value="cjardim"/>
  </inferenceS>
</context>
```

If a user wants to create a group (Figure 1(c)) — or else be included in an existing group — *WebRegister* checks both if the current user and the current group are registered with the CK database. If it succeeds, *WebRegister* calls the GetInverse service that checks for the existence of some *what* inference where the type equals "group" and value equals the name of the group given (Example 5).

```
<!--            Example 5            -->
<context>
  <inferenceS>
    <inference dimension="what" type="group" value="intermedia"/>
  </inferenceS>
</context>
```

To include a member in a group, *WebRegister* calls the PutRules service to register the corresponding premise and *what* inference as illustrated in Example 6.

```
<!--            Example 6            -->
<context>
  <premiseS>
    <premise dimension="who" type="login" value="cjardim"/>
  </premiseS>
  <inferenceS>
    <inference dimension="what" type="group" value="intermedia"/>
  </inferenceS>
</context>
```

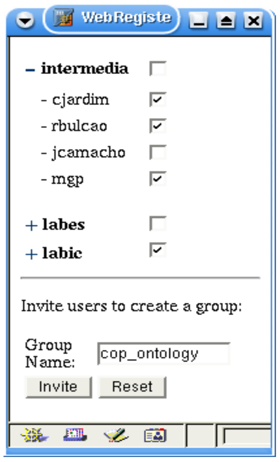
An important feature for CoP/CSCW services is the ability for users to create a group based on the location information of the users registered with CK. An application such as *WebRegister* can call CK to obtain a list of users that, belonging to a given group, have registered themselves giving the same location. The idea is to be able to create CoP based on the whole group (e.g. users interested in the Rain Forest), but constrained to those that informed their location to a common place (e.g. those attending a conference in Salvador).

Figure 2 depicts the *WebRegister* application with a list of laboratories and their respective users registered with CK. CoP may be explicitly created by selecting its participants — even users from different places (e.g. the "labic" and "intermedia" laboratories) — with a common interest, "ontology", in this case.

4.3. The WebMemex application: exploiting groups

Several web-based recommendation systems have been proposed in the literature, the essence in many cases being that members of a group are well-suited to recommend each other appropriate references. As an example, ReachOut [Ribak et al., 2002] is a system that supports users in interested groups to explicitly ask recommendation to other group members. In another original approach, the underlying network of referencing in research papers is used to support building a system to recommend citation of research papers [McNee et al., 2002].

WebMemex is an application that recommends web pages related to the one the user is currently visiting. It was developed on top of a high level-architecture that provides



Invite users to create a group:

Group Name: cop_ontology

Figure 2: A community of practice called "cop_ontology" is created by choosing all users from the "labic" laboratory and some from the "intermedia" one.

capture, linking, storage, retrieval and access capabilities [Macedo et al., 2003]. *WebMemex* captures and recommends web pages for groups of users by means of its own web proxy server. In order to identify users and group memberships, the application relies on the fact that the web browser client must establish a connection to that web proxy server for each HTTP request. The interface of *WebMemex* is depicted in Figure 3.

When using *WebMemex*, users need to be aware of the set of groups they belong to. Knowing who the current user is, *WebMemex* calls the GetAny service which searches for all (last value = "ALL") *what* inferences which type is group and a premise with information that matches the current username, as shown in Example 7.

```
<!--            Example 7            -->
<last value="ALL"/>
<context>
  <premiseS>
    <boolean type="AND">
      <premise dimension="who" type="login" value="cjardim"/>
    </boolean>
  </premiseS>
  <inferenceS>
    <inference dimension="what" type="group"/>
  </inferenceS>
</context>
```

As a result, CoP can be created based on the location information of users being in a range of location defined by the application. In a research scenario, people with common interests, but from different laboratories, can recommend web pages among themselves by creating a group via *WebRegister*. This application creates a list with all users registered and their respective locations (*who*+*where* information). From that list, a user can choose the members that will be invited to participate in CoP. Therefore, the updated group information in the *WebMemex* application (drop-down box) allows users to exchange web pages information.

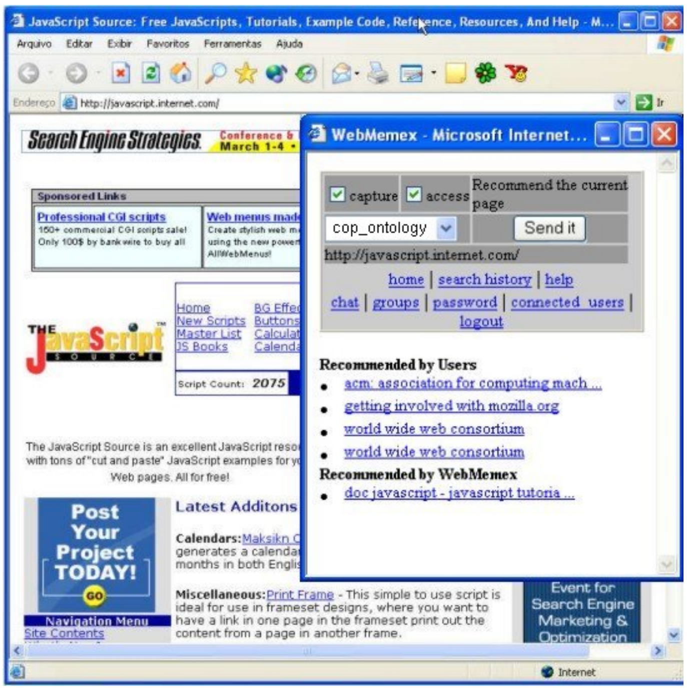

WebMemex - Microsoft Internet...
capture
access
Recommend the current page
cop_ontology
Send it
http://javascript.internet.com/
home | search history | help
chat | groups | password | connected users | logout
Recommended by Users
acm: association for computing mach ...
getting involved with mozilla org
world wide web consortium
world wide web consortium
Recommended by WebMemex
doc javascript - javascript tutoria ...

Figure 3: The WebMemex interface (foreground): the drop-down box corresponds to the groups a user belong to, and may make use of both location information and group membership information, as demanded to support CoP.

5. Concluding remarks

We have shown how the Context Kernel infrastructure can be used to manage context information of CoP/CSCW applications, as demonstrated by *WebMemex*. As a conclusion, we advocate that the Context Kernel Web Service is a viable approach for managing context information of CoP/CSCW applications since:

1. context information can be shared among applications such as user, group and location information to provide group awareness;
2. Context Kernel allows to integrate applications that have not been designed to work together (e.g. *WebMemex*).

Most efforts reported on the CSCW literature are broader than the current version of the Context Kernel in terms of the amount, type and depth of the CSCW-related functionalities provided. The main contribution of the Context Kernel is relative to its proposal of a Web Service-related standards as the model for communication as well as the sharing of social awareness context information; the emphasis on social awareness is due to the expected delays associated with using the Web as the communication infrastructure. The implications include: (i) the support to several levels of heterogeneity among applications; (ii) applications can not only store, but also share context information and (iii) components can provide independent but integrated distribution of responsibilities inter and intra applications.

It is important to observe that the underlying model of the Context Kernel demands that the applications define what information they store and what information — stored by other applications — they obtain from the repository. This means that users are unlikely to be in charge of specifying rules for the impact of some context information — the approach is that the conceptual model of the applications take into account the existence of such a repository.

Until now, one limitation of our approach is about membership of users in a temporary permanent mode. Temporary membership in a group is achieved while users are synchronously logged in and can be exploited by applications that connect people in a given neighborhood. Our approach deals with permanent membership; it is achieved by adding users to a group when they register their location, but not removing them from the group when they log off that location.

Another contribution results from the specification of the context information based on the classical dimensions for context: *who*, *when*, *where*, *what*, *why* and *how*. Although the verbosity of the model can be a problem in terms of processing and long-term persistent storage, we have been studying the possibility of incorporating, in the long term, artificial intelligence approaches on the server side (see [Bulcão Neto and Pimentel, 2003]) so as to benefit many CoP/CSCW applications.

Acknowledgments

We would like to thank to FAPEMA (03/345), FAPESP and CNPq funding agencies for supporting this research. We would also like to thank to Alessandra Macedo and Khai Truong for giving us the opportunity of use of the *WebMemex* application, to Renan Cattelan for his helpful comments, and to Carlos Arruda Jr for his initial work on the Context Kernel Web Service.

References

Abowd, G., Mynatt, E. D., and Rodden, T. (2002). The Human Experience. *IEEE Pervasive Computing*, 1(1):48–57.

Agostini, A., Albolino, S., Boselli, R., De Michelis, G., De Paoli, F., and Dondi, R. (2003). Stimulating Knowledge Discovery and Sharing. In *International ACM SIGGROUP Conference on Supporting Group Work*, pages 248–257, Sanibel, Florida. ACM Press.

Anderson, G. E., Graham, T. C. N., and Wright, T. N. (2000). DragonFly: Linking Conceptual and Implementation Architectures of Multiuser Interactive Systems. In *Proceedings of the ACM Conference on Software Engineering*, pages 252–261.

Arruda Jr, C. R. E., Bulcão Neto, R. F., and Pimentel, M. G. C. (2003). Open Context-aware Storage as a Web Service. In *Proceedings of the International Workshop on Middleware for Pervasive and Ad-Hoc Computing held as part of the ACM/IFIP/USENIX International Middleware Conference*, pages 81–87.

Arsanjani, A., Hailpern, B., Martin, J., and Tarr, P. (2003). Web Services: Promises and Compromises. *ACM Queue (Web Services)*, 1(1):48–58.

Bulcão Neto, R. F. and Pimentel, M. G. C. (2003). Interoperabilidade Semântica entre Aplicações Cientes de Contexto. In *Anais do Simpósio Brasileiro em Sistemas Multimídia e Web (WebMídia)*, pages 371–385.

Burner, M. (2003). The Deliberate Revolution. *ACM Queue (Web Services)*, 1(1):28–37.

Chinnici, R. et al. (2004). Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, Working Draft. Available online in `http://www.w3.org/TR/wsdl20/`.

Congar, T., Noyes, J., and Kimble, C. (1999). CLIMATE: A framework for developing holistic requirements analysis in virtual environments. *Interacting with computers*, 11:387–402.

Dey, A. K. (2001). Understanding and Using Context. *Personal and Ubiquitous Computing Journal*, 5(1):4–7.

Dey, A. K., Salber, D., Abowd, G. D., and Futakawa, M. (1999). The Conference Assistant: Combining Context-Awareness with Wearable Computing. In *Proceedings of the International Symposium on Wearable Computing*, pages 21–28.

Fallside, D. C. (2001). XML Schema Part 0: Primer, W3C Recommendation. Available online in `http://www.w3.org/TR/xmlschema-0/`.

Fitzpatrick, G., Mansfield, T., Kaplan, S., Arnold, D., Phelps, T., and Segall, B. (1999). Augmenting the Workaday World with Elvin. In *Proceedings of the European Conference on Computer-Supported Cooperative Work*, pages 431–450.

Fuchs, L. (1999). AREA: A Cross-Application Notification Service for Groupware. In *Proceedings of the European Conference on Computer-Supported Cooperative Work*, pages 61–80.

Garlan, D., Siewiorek, D., Smailagic, A., and Steenkiste, P. (2002). Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31.

Gross, T. and Prinz, W. (2003). Awareness in Context: a Light-Weight Approach. In *Proceedings of the European Conference on Computer-Supported Cooperative Work*, pages 295–314.

Laurillau, Y. and Nigay, L. (2002). Clover Architecture for Groupware. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pages 236–245.

Lave, J. and Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation*. Cambridge University, New York, NY.

Macedo, A. A., Truong, K. N., Camacho-Guerrero, J. A., and Pimentel, M. G. C. (2003). Automatically Sharing Web Experiences through a Hyperdocument Recommender System. In *Proceedings of the ACM Conference on Hypertext and Hypermedia*, pages 48–56. ACM Press.

Mack, R., Ravin, Y., and Byrd, R. J. (2001). Knowledge portals and the emerging digital knowledge workplace. *IBM Systems Journal*, 40(4):925–955.

McNee, S. M., Albert, I., Cosley, D., Gopalkrishnan, P., Lam, S. K., Rashid, A. M., Konstan, J. A., and Riedl, J. (2002). On the Recommending of Citations for Research Papers. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pages 116–125.

Millen, D. R. and Fontaine, M. A. (2003). Improving Individual and Organizational Performance through Communities of Practice. In *International ACM SIGGROUP Conference on Supporting Group Work*, pages 205–211. ACM Press.

Mitra, N. (2003). Simple Object Access Protocol (SOAP) 1.2, W3C Recommendation. Available online in `http://www.w3.org/TR/soap12-part0/`.

Nonaka, I. and Takeuchi, H. (1995). *The Knowledge Creating Company*. Oxford University Press.

Patterson, J. F., Day, M., and Kucan, J. (1996). Notification Servers for Synchronous Groupware. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pages 122–129.

Prinz, W. (1999). NESSIE: An Awareness Environment for Cooperative Settings. In *Proceedings of the European Conference on Computer-Supported Cooperative Work*, pages 391–410.

Ribak, A., Jacovi, M., and Soroka, V. (2002). Ask Before You Search: Peer Support and Community Building with ReachOut. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pages 126–135.

Rittenbruch, M. (1999). Atmosphere: Towards Context-Selective Awareness Mechanisms Human-Computer Interaction: Communication, Cooperation and Application Design. In *Proceedings of the International Conference on Human-Computer Interaction*, pages 328–332.

Roberts, T. L. (1998). Are newsgroups virtual communities? In *Proceedings of CHI´98*, pages 360–367.

Salber, D., Siewiorek, D. P., and Smailagic, A. (2001). Supporting Mobile Workgroups on a Wireless Campus. In *Proceedings of the International Workshop on Human Computer Interaction with Mobile Devices*.

W3C (2002). Web Services Activity. Available online in `http://www.w3.org/2002/ws`.