A Context Model for Personal Knowledge Management

Sven Schwarz

`Sven.Schwarz@dfki.de`
German Research Center for Artificial Intelligence (DFKI GmbH)
D-67608 Kaiserslautern, Germany

Abstract. In the research project EPOS[1] we build a pro-active, context-sensitive support system to aid the user with his knowledge work, which is mostly about searching, reading, creating, and archiving of documents. In order to avoid distracting the user, the context gathering is realized by installable user observation plugins for standard applications such as Mozilla Firefox and Thunderbird.

The main part of this paper is about the definition of a context model for the personal knowledge management domain. The context model incorporates only contextual elements relevant to satisfy the knowledge worker's potential information need. It stores only information items known to the user (such as links to his own documents, folders, etc.), as well as, shared ontologies to assure an understanding of the context.

The context is modeled in RDF/S and can be retrieved by context-aware applications from the context support system via an XML-RPC call.

1 Introduction

Although the same term is used by linguists, psychologists, and computer scientists, "context" is understood in a variety of ways. To talk about context does not make sense without talking about its application and the scenario it is used in. We will therefore start with a short introduction of the research scenario.

The overall goal of the research project EPOS [4] (Evolving Personal to Organizational knowledge Spaces) is to build up *organizational/group-wide* structured knowledge from the information items and structures (already) present at the members of the organization/group. In order to encourage each worker to archive and structure his own knowledge, the worker himself should be the first person to utilize his own structuring effort. This is the knowledge management *bait*. Instead of doing it because of an obligation, the worker does it to enhance his own support, to make his own work faster and better.

Now, what does it mean to support a worker with his own structures and information items? There are two alternatives for potential support: First, a sophisticated retrieval of structures (folders/categories) and information items

(documents) will support the user when he searches for material. Second, according to the worker's current context, potentially relevant structures and information items can be presented in a pro-active way, i. e., without the worker even asking for them.

Without going into discussions about good, humane assistance interfaces, the second alternative demands for capturing the worker's context as unobtrusive as possible. EPOS relies on user observation to gather evidences for contextual information.

The remaining of this paper is structured as follows: Section 2 describes our scenario and the knowledge worker's world. Section 3 defines an appropriate context model for that scenario. Section 4 then explains how the context model is being filled. Section 5 shows how the context can be retrieved and used. After some related work in section 6, the paper concludes with section 7.

2 The EPOS Scenario

A knowledge worker searches for documents, reads, writes, archives, and structures (classifies) them. A lot of the work is already done with the standard functionality of today's PCs, that is, by using the functionality of standard applications (web browser, email client, text processor) and the operating system (file system). Structuring/classification of documents is, for instance, typically done using file folders. However, classification using file folders poses two problems: First, a document can only be classified using one folder, and second, folder hierarchies nearly never follow a consistent scheme. Therefore, EPOS initially starts with folder hierarchies already present on the user's PC. Beyond this, EPOS allows to build up additional, more consistent container hierarchies, which can then be used for a more consistent and multiple classification of documents.

A user will typically create several such container hierarchies, resembling his individual and subjective views of the world. Consequently, these container hierarchies are also called *views*. One view can be the *topics* view containing a hierarchy of containers representing scientific topics. To be more precise, such a view is a DAG (directed acyclic graph), so a topic can have more than just one super topic. That way the user creates a simple topic ontology. Analogously, the user can also build a *projects* view including containers for relevant projects. It can also be useful to have a *contacts* view, and maybe an *events* view, too, and so on. The point is, a document can be classified by putting it into *all* relevant containers. Let's take this paper as an example: It is about context and user observation, and it has to do with project EPOS. Hence, we can put that paper into the following three containers (see figure 1):

- `Topics/KnowledgeManagement/Context`
- `Topics/MachineLearning/UserObservation`
- `Projects/EPOS`
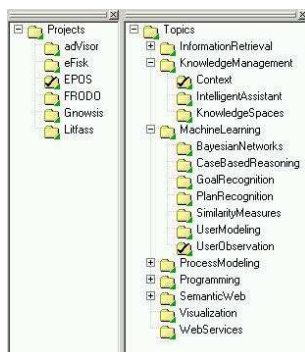
Fig. 1. Container hierarchies resemble the user's individual *views*, for example, relevant `Projects` and `Topics`.

2.1 The Personal Knowledge Space

EPOS makes use of and accesses the *native structures* present on the user's PC. These are the documents (files), folders, emails, and so on. We pointed out, that EPOS also promotes building up multiple and more consistent container hierarchies (views). Documents can be put into these containers as a form of classification. It is important to note, that for the personal knowledge space these view-containers do not simply *contain* some documents. Moreover, the *semantics* of the container is defined by the contained documents (instance-based semantics). That way, EPOS can help to classify both old and new documents.

Moreover, EPOS allows for relating several resources with each other. A document can be linked to some other document via a `dc:relation` link for example. Even more interesting is the usage of links with richer semantics: Using `dc:creator` some web page can be linked to some person (address book contact), declaring this person is the creator of that web page (see figure 2). To be more precise, EPOS supports building up a *personal semantic web* on the user's PC. This approach has been described in [10, 11].

2.2 The Knowledge Worker's World

Let's sum up the facts about our scenario by defining the (closed) world of a knowledge worker by giving a concrete list of the objects and tools he uses:

Document-like objects:

- Documents (HTML, plain text, wiki pages, PDF)
- Emails
- Notes (tiny text snippets created with EPOS Notes, a proprietary notes tool)
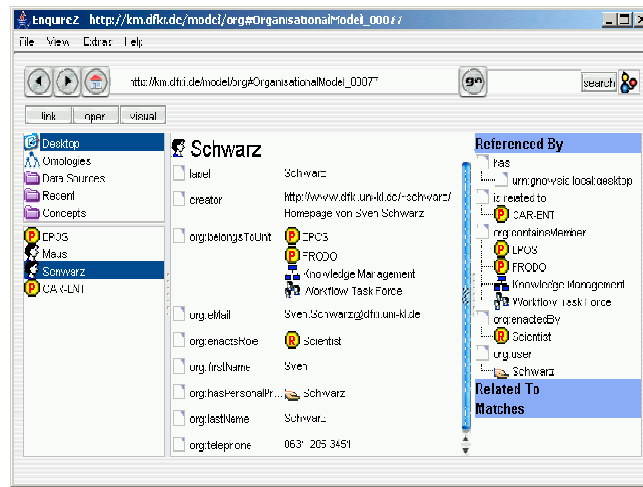
Fig. 2. EPOS allows for creating and browsing the personal semantic web. Example: The resource `Sven Schwarz` is an organizational person with some attributes and some links. Some attributed link (`dc:creator`) shows, he is the creator of his homepage.

Objects useful for structuring/classification:

- File-Folders
- Topics (knowledge management, machine learning, etc.)
- Contacts, addresses (persons)
- Organizational entities (projects, organizations, etc.)
- Workflows and Tasks (from a workflow management system)

Note: All these structuring objects are potentially available both as individual ones (stored on the user's PC), as well as, organizational ones (hosted on some group-wide server) [5].

Applications:

- Text processor
- Web browser
- Email client
- Address book (of the email client)
- EPOS Notes (proprietary, tiny notes tool, which is part of EPOS)
- File system, brainFiler[2] (classification of documents)
- Organizational Repository (manages organizational entities, not handled in detail in this paper)
- FRODO TaskMan (Workflow Management System [6, 1])

Now, that we have explained the scenario and defined the (abstract) world of the knowledge worker, we can go over to explicitly model his context.
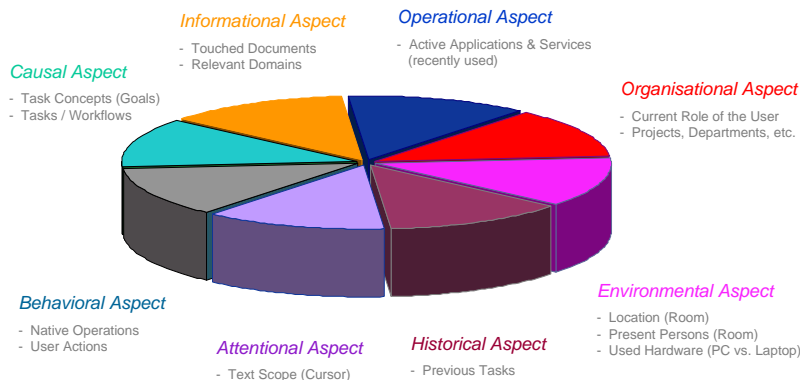
---
[2] `http://www.brainbot.de`

Fig. 3. In EPOS, the user's context comprises various aspects.

3 Modeling Context

When modeling the user's context we have to take into account his working (desktop) environment. In our scenario, a knowledge worker is mainly doing document oriented work (searching, reading, creating, archiving, classifying), as well as, organizational work (communicate with colleagues, executing workflow tasks). Hence, the user's context contains information about *currently or recently* read documents, relevant topics, relevant persons, relevant projects, etc., that is, exactly the objects listed in section 2.2. In EPOS, the user's context comprises a variety of aspects (see figure 3) each containing aspect-specific contextual elements. Examples in this paper focus on the informational and the organizational aspect.

We used Protégé [3] to realize an explicit and formal context model in RDF/S [4]. The classes, especially the `Context` class, and properties therein define the parameters for the user's context. An *actual* user context is represented by an instance of this context model, respectively, of the `Context` class. This instance, simply called *context object*, is permanently kept up-to-date. Hence, the *current* context of the user is then just a snapshot of the (one and only) context object and its contents. For every contextual information available, the context object contains so-called *contextual elements* wrapping the respective contextual information. For example: There could be some contextual elements wrapping the links to web pages the user recently browsed to while other contextual elements wrap some of the user's topics currently relevant.

Besides the core contextual information, these contextual elements also contain information about their confidence. Observed contextual information will have a confidence of `1.0` whereas estimated contextual elements, like automati-

---

[3] `http://protege.stanford.edu/`

[4] `http://www.w3.org/TR/rdf-schema/`

cally classified topics, will mostly have a lower confidence. Furthermore, contextual elements contain information about their creation and the support they give to or receive from other contextual elements. This allows for some explanation, as well as, for some global probability calculation on all of the elements in the user's context.

More important than the structure of the contextual element is the contextual information they convey. Before applications can use a context snapshot they have to *understand* the contextual information. As our context model is grounded in the user's personal knowledge space (see section 2.1), contextual elements represent real-life entities of the user's world. For example: touched documents, visited/classifying folders, recently touched or related persons, projects, and so on. This makes it easy for applications to understand and use the contextual information provided by our context service. Figure 5 shows a concrete real-life snapshot of the user's context. Only the informational and the organizational aspect have been extracted, e. g., behavioral information (actions of the user) has been elided to keep the RDF data as readable as possible.

It is important, that the context relies on the user's own personal knowledge space, i. e., his *individual and subjective* view of the world; therein for example the previously described topic ontology. This holds especially for the similarity of two user contexts. Their similarity is grounded on the similarity of the user's own view of the world. For example: For one user the topics `workflow` and `business process modeling` are the same while for a different user both topics are only similar. This means, the context model and contextual similarity adapt to the user.

4 Gathering and Eliciting Context

Context can be gathered and modeled using user observation and/or user feedback. However, relying on user feedback is critical. The context-sensitive support must be nearly perfect to keep even motivated users doing feedback all of the time. But still, even with perfect support, the user will stop giving feedback as soon as he is feeling stress and has no more time "to feed his context tamagotchi". Thus, *unobtrusive* and *reliable* gathering and modeling of context demands for permanent user observation. Of course the user must be able to toggle the user observation on and off whenever he likes.

Instead of integrating some low-level hooks into the operating system, EPOS created plugins for a set of standard applications from the scenario. The advantage is, that the plugins integrated in the applications can deliver more precise information than raw mouse movements, clicks, or key strokes. The price we pay is, that new applications can not be observed unless someone writes an observation plugin for it.

Again our scenario tells us, which applications we need to observe to gather the most interesting elements of the user context:

– File Explorer (handling documents and folders)
– Email Client (communication with colleagues)

– Web Browser (searching and browsing for documents/information)
– Text Processor (reading and writing documents)

User Observation for Mozilla Thunderbird (email client) and Mozilla Firefox (web browser) is realized in form of Mozilla extensions. That way they can be easily installed using an XPI-file. Besides, the user can configure and (de)activate the observation from within the application. File explorer and text processor plugins are still in development, but email communication and web browsing already allows for gathering very interesting contextual information. Anyway, all plugins send the observed user operations via simple XML-RPC calls to the listening *Context Service*, where the context object is updated accordingly. For example, when I opened an email in Mozilla Thunderbird, the following XML-RPC / HTTP POST is sent to `http://localhost:9998/RPC2`:

```
<methodCall>
  <methodName>epos_context_MozUserObsApi.eventViewEmail</methodName>
  <params><param><value><struct>
   <member><name>EmailURI</name>
     <value><string>imap://schwarz@serv-4100/INBOX#18423</string></value>
   </member><member><name>FolderURI</name>
    <value><string>imap://schwarz@serv-4100/INBOX</string></value>
   </member><member><name>Subject</name>
    <value><string>Personal and [...] - New Issue Alert</string></value>
   </member><member><name>Sender</name>
    <value><string>"alerts@springerlink.de</string></value>
   </member>
   ...
  </struct></value></param></params>
</methodCall>
```

Note, that the parameters are enclosed by `<![CDATA[...]]>`, really. For simplicity, these tags have been removed in the snippet above. After the XML-RPC has been sent, a servlet in the Context Server catches this event and calls the corresponding method:

```
public void eventViewEmail(Hashtable hashtable) {
  String emailURI = (String)hashtable.get("EmailURI");
  String subject = (String)hashtable.get("Subject");
  ...
  ViewEmail nop = new ViewEmail();  // nop stands for "native operation"
  nop.setURI(emailURI);
  nop.setSubject(subject);
  ...
  getContextServer.addNopToContext(nop);
}
```

We created an ontology of user operations using Protégé and RDF/S. The Java class `ViewEmail` used in the code snippet above has been generated by `rdf2java` [2] to wrap the corresponding RDF class in the Context Model's RDF Schema. All properties (outgoing edges) of some RDF object can be get and
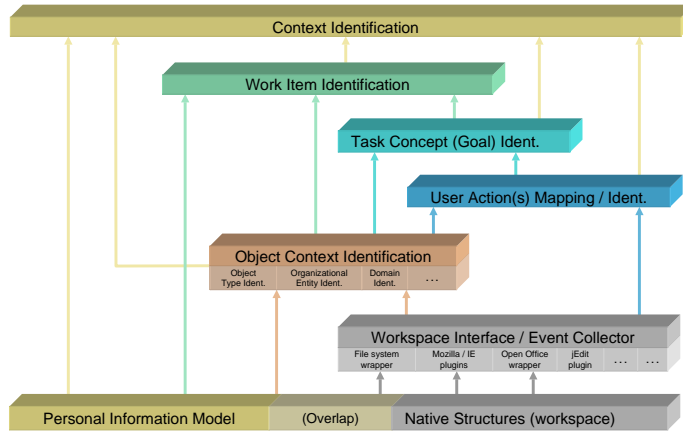
Fig. 4. The context elicitation is realized as a modular, pipelined architecture.

set by the Java wrapper. This allows very comfortable creation, handling, and passing of RDF data.

Besides parameterizing, the user operations are modeled hierarchically (`is-a` relations) to allow for fine, as well as, coarse handling of user operations. For example: `ViewEmail` and `SendEmail` are `EmailOperation`s; and `AddBookmark` is both a `BookmarkOperation` and an `ArchiveOperation`. A context-aware application can decide wether it is sufficient to know about the user doing some `EmailOperation` or wether `ViewEmail` and `SendEmail` leads to different user contexts and, thus, to different contextual support.

Adding the new *native operation* (*nop*) to the context means, first, creating a *ContextualElement* for the nop, and, second, raising a *ContextualElementAdded-Event*. This event will be catched by context elicitation modules. Taking the received user operations as evidences, more higher-level contextual information is estimated: Potentially relevant topics the user works on are inferred on the basis of recently touched documents (see section 2). Additionally, a case-based reasoning approach [12] estimates potential user goals and relevant workflow tasks. Figure 4 shows the context elicitation modules working in a pipelined architecture.

5   Retrieval and Usage of Context

In EPOS, context gathering and elicitation is realized as an autonomous, optional service. That service is used in two ways: *Context-Polling* and *Context-Listening*.

Context-Polling: Requesting a snapshot of the user's current context or a specific part of it. This is often triggered by some user function. The context can then be used, for instance, to to show him recent or relevant items at that specific

time. Another example: The user wants to save a downloaded document. The current context can be used to propose a corresponding directory. Furthermore, if the document is annotated with the context, the document can be retrieved via a context-enabled query later. From inside the EPOS framework, the context model can be retrieved simply by calling

```
Model model = ContextService.getContextApi().getSnapshot();
```

From outside the EPOS framework, or from non-Java code, the same can be achieved by a simple XML-RPC call. Here is an example of how to request a context snapshot from some Java application – the result will an output similar to that in figure 5:

```
import org.apache.xmlrpc.XmlRpcClient;
...
XmlRpcClient c = new XmlRpcClient("localhost", 9998);
String method = "epos_context_ContextApi.getSnapshotAsRdfXml";
java.util.Vector params = new java.util.Vector();
Object result = c.execute(method, params);
System.out.println("context as RDF/XML:\n" + result);
```

Context-Listening: If some application/service registers at the Context Service as a listener, it will be informed about some specific change (event) in the user's context. On one hand, this is heavily used by the context elicitation modules. They are triggered as soon as new contextual elements enter the context. On the other hand, this is needed to realize *pro-active* support. If the user shifts his actions towards some context for which relevant support is available, this support can be presented to the user without him having to asking for it.

EPOS uses this method to trigger and supply a so-called *EPOS Assistant Bar*, which presents currently relevant documents, topics, organizational entities (persons, projects) and workflow tasks to the user. The context listening can also be used to trigger some context-sensitive help system for instance. By now, the context listening protocol is quite proprietary and can only be used by modules *inside* the EPOS framework, but extending event listening to XML-RPC clients is just a minor, technical issue.

6   Related Work

The Lumiere project [9] realized pro-active assistance for computer software users. A bayesian user model was employed to infer a user's information needs. While Lumiere mainly focussed on enhancing the usage of some software, EPOS pursues assisting the users' work in general. That means, first, not only one application is observed and enhanced with some (closed) assistance, and second, not only the user's *information need* is elicited, but his context.

Watson [3] observes user interaction with everyday applications and attempts to automatically fulfil a user's information need by querying common Internet information sources automatically. Analogously to Watson, EPOS uses a *Content Analyzer* to automatically classify some viewed document and estimate

contextually relevant topics that way. Furthermore, EPOS also realizes a real task-specific information delivery using the information stored in a workflow management system. It knows about which documents are relevant for which task. As the context also contains relevant workflow tasks, the corresponding documents can be presented to the user [8].

Fenstermacher [7] envisions revealing and storing process-relevant information by automatically classifying documents the user touches during his work. For this purpose low-level observation hooks are installed directly into the operating system. EPOS relies on observed user actions with higher semantics, like `SendEmail`, however, collaborating with Fenstermacher we may integrate these low-level hooks to be able to observe applications without observation plugins (see section 4).

Zacarias et al [13] introduce an operating systems metaphor, where the user's *operating system* acts as an *engine* managing the execution of pre-defined flows of work. Besides that very interesting metaphor, the authors carry out a case study where students observe and record the actual tasks, actions, and interactions of persons at work. EPOS will keep up collaboration with Zacarias' team to enhance the context model according to their observations.

7   Conclusion

We have presented an explicit and formal model of a knowledge worker's context. The user's context, being an instance of this context model, is fed by user observation plugins installed in standard applications like Mozilla Thunderbird and Mozilla Firefox. The context relies on the user's personal workspace, i.e., touched or elicited resources in the context are mainly the user's own files, folders, email contacts, and so on. This means, that the contextual elements are known entities and, hence, the context can be easily understood and used. It also means, that the elicited context adapts to the specific user at a specific time. Even if the user changes his working procedures and domain over time, the context will adapt automatically, because according changes on the user's PC will take place: new folders and new documents emerge, re-classification of documents occurs, etc.

The set of supported applications and hence the set of observable user actions is limited by the set of available user observation plugins. Implementing new plugins for yet unsupported applications is merely a question of how to integrate plugins into that application and how to recognize the the user's actions therein.

An application can act context-aware by either explicitly requesting a context snapshot or by registering itself as a listener to context changes. The context service currently elicits *projects*, *persons*, and *topics* potentially relevant due to text content created or viewed by the user. Ongoing research will cover the elicitation of higher-level contextual information such as the user's *goals* or relevant *workflow tasks*.

That research will be followed by an evaluation of the fitting and utility of the elicited context. Furthermore, the evaluation will have to show whether and how much the users really like and use the context-aware support.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:context="http://km.dfki.de/context#"
    xmlns:domain="http://km.dfki.de/domain#"
    xmlns:org="http://km.dfki.de/org#"
    xmlns:object="http://km.dfki.de/nasti/objects#">
  <context:Context>
    <context:informationalAspect>
      <context:InformationalAspect>
        <context:contains>
          <object:EMail rdf:about="imap://schwarz@serv-4100/INBOX/;UID=17050">
            <object:subject>paper on data integration framework, ISWC 2005</object:subject>
            <object:recipients>Sven Schwarz &lt;schwarz@dfki.uni-kl.de&gt;</object:recipients>
            <object:sender>Leo Sauermann &lt;leo@gnowsis.com&gt;</object:sender>
            <object:content>Hi Sven! [...] I would like to write a paper about [...]
                            semantic web and desktop applications [...]
            </object:content>
            <object:lastAccess>2005-04-13T14:45:22</object:lastAccess>
            <context:confidence>1.0</context:confidence>
          </object:EMail>
        </context:contains>
        <context:contains>
          <object:HtmlFile rdf:about="http://jena.sourceforge.net/">
            <object:location>http://jena.sourceforge.net/</object:location>
            <object:title>Jena Semantic Web Framework</object:title>
            <object:fileType>text/html</object:fileType>
            <object:lastAccess>2005-04-13T14:45:42</object:lastAccess>
            <context:confidence>1.0</context:confidence>
          </object:HtmlFile>
        </context:contains>
        <context:contains>
          <domain:DomainConcept rdf:about="urn:brainfiler:dfkiklkm:schwarz:Category:136">
            <domain:name>Gnowsis</domain:name>
            <context:confidence>0.717095</context:confidence>
            <context:supportedBy rdf:resource="http://jena.sourceforge.net/"/>
            <context:supportedBy rdf:resource="imap://schwarz@serv-4100/INBOX/;UID=17050"/>
          </domain:DomainConcept>
        </context:contains>
        <context:contains>
          <domain:DomainConcept rdf:about="urn:brainfiler:dfkiklkm:schwarz:Category:105">
            <domain:name>RDF[S]</domain:name>
            <context:confidence>0.350096</context:confidence>
            <context:supportedBy rdf:resource="http://jena.sourceforge.net/"/>
          </domain:DomainConcept>
        </context:contains>
      </context:InformationalAspect>
    </context:informationalAspect>
    <context:organizationalAspect>
      <context:OrganizationalAspect>
        <context:contains>
          <org:Person rdf:about="mailto:leo@gnowsis.com">
            <org:eMail>leo@gnowsis.com</org:eMail>
            <org:firstName>Leo</org:firstName>
            <org:lastName>Sauermann</org:lastName>
            <context:confidence>1.0</context:confidence>
            <context:supportedBy rdf:resource="imap://schwarz@serv-4100/INBOX/;UID=17050"/>
          </org:Person>
        </context:contains>
      </context:OrganizationalAspect>
    </context:organizationalAspect>
  </context:Context>
</rdf:RDF>
```

Fig. 5. Snapshot of the user's context. Only the informational and the organizational aspect have been extracted.

References

1. Homepage of FRODO TaskMan: `http://www.dfki.de/frodo/taskman/`.
2. Homepage of rdf2java: `http://rdf2java.opendfki.de/`.
3. Jay Budzik and Kristian J. Hammond. Watson: An infrastructure for providing task-relevant, just-in-time information.
4. Andreas Dengel, Andreas Abecker, Jan-Thies Bähr, Ansgar Bernardi, Peter Dannenmann, Ludger van Elst, Stefan Klink, Heiko Maus, Sven Schwarz, and Michael Sintek. Evolving Personal to Organizational Knowledge Spaces. Project Proposal, DFKI GmbH Kaiserslautern, 2002.
5. Ludger van Elst and Andreas Abecker. Integrating Task, Role, and User Modeling in Organizational Memories. In *14 Int. FLAIRS Conference, Special Track on Knowledge Management, Key West, Florida, USA*, May 2001.
6. Ludger van Elst, Felix-Robinson Aschoff, Ansgar Bernardi, Heiko Maus, and Sven Schwarz. Weakly-structured workflows for knowledge-intensive tasks: An experimental evaluation. In *IEEE WETICE Workshop on Knowledge Management for Distributed Agile Processes: Models, Techniques, and Infrastructure (KMDAP03)*. IEEE Computer Press, 2003.
7. Kurt D. Fenstermacher. Revealed Processes in Knowledge Management. In Klaus-Dieter Althoff, Andreas Dengel, Ralph Bergmann, Markus Nick, and Thomas Roth-Berghofer, editors, *3rd Conference on Professional Knowledge Management – WM 2005*, pages 397–400. DFKI GmbH, 2005.
8. Harald Holz and Frank Maurer. Knowledge management support for distributed agile software processes. In *Advances in Learning Software Organizations, 4th International Workshop, LSO 2002, Chicago, IL, USA, August 6, 2002, Revised Papers.*, volume 2640. Springer, 2002.
9. E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 256–265, Madison, WI, July 1998.
10. Leo Sauermann. The gnowsis-using semantic web technologies to build a semantic desktop. Diploma thesis, Technical University of Vienna, 2003.
11. Leo Sauermann and Sven Schwarz. Introducing the gnowsis semantic desktop. In *Proceedings of the International Semantic Web Conference 2004*, 2004.
12. Sven Schwarz and Thomas Roth-Berghofer. Towards goal elicitation by user observation. In *Proceedings of the FGWM 2003 Workshop on Knowledge and Experience Management*, Karlsruhe, 2003.
13. Marielba Zacarias, Artur Caetano, H. Sofia Pinto, and Jose Tribolet. Modeling contexts for business process oriented knowledge support. In *Proceedings of the WM 2005 Workshop on Knowledge Management for Distributed Agile Processes (KMDAP 2005)*, Kaiserslautern, 2005.