ABSTRACT

We present a novel approach for proactive support of user in knowledge intensive organisations. Whilst once information was a scarce resource, nowadays all kinds and qualities of information are available. However human attention has become a scarce resource which is difficult to manage and support. Our attention management system proactively supports the user in dealing with processes, activities and tasks defined by a semantically-enhanced business workflow. Moreover the user is supported in reacting on changes respecting the users’ context and preferences. The approach is based on an expressive attention model, which is realized by combining context-aware ECA rules with ontologies and an appropriate preference model. We present the system’s architecture, describe its main components and present early evaluation results. Our system is particularly deployed in a use case related to eGovernment. Nevertheless the architecture we present is general, and may be used in all kind of information and knowledge systems where handling the user attention is of an important interest.

Keywords
Attention Management, Semantic eGovernment, Semantic Technologies, ECA rules

1. INTRODUCTION

Success factors in knowledge-intensive and highly dynamic business environments include the ability to rapidly adapt to complex and changing situations and the capacity to deal with large quantities of information of all sorts. For knowledge workers, these new conditions have translated in acceleration of working performance, multiplication of projects in which they are involved and increased collaboration with colleagues, clients and partners. Knowledge workers are overloaded with potentially useful and continuously changing information originating from a multitude of sources and tools. A significant part of a knowledge worker’s day can be occupied with searching and looking for information. Moreover, they have to interact with various people from different organizations and in different contexts. In addition, software tools such as email and instant messaging have added to the whole complexity and have created new interruption channels for knowledge workers.

In order to cope with changes of the business environment, the attention of knowledge workers must be always paid on the most relevant information sources. Indeed, a basic requirement of knowledge workers is to be up-to-date with information while facing an information overload situation. In other words, the issue is how to select those information resources whose reading will give most benefits to the reader. Moreover, agility and proactive computer can be useful in an environment of knowledge workers with overburdened memories: The computer should know what a knowledge worker works with and show him/her relevant information before they need them, in a kind of pre-search function.

According to Davenport and Beck [6], attention is defined as a “focused mental engagement on a particular item of information”. According to [4], attention is a process of selection and selective processing, required because the brain has a limited information processing capacity. In an enterprise context, attention management refers to supporting knowledge workers focus their cognitive ability on a particular organizational task and on the information resources required to accomplish it. In particular support is required for searching, finding, selecting and presenting the most relevant and up-to-date information without distracting workers from their activities. Information retrieval systems have provided means for delivering the right information at the right place and in right time. The main issue with existing systems is that they do not cope explicitly with the information overload, i.e. it might happen that a knowledge worker “overlooks” important information.

The goal of attention management systems is to avoid information overload and to provide notifications about new and changed, relevant information [6]. Moreover, the frequently changing environment requires not only very effective systems for alerting knowledge workers that some relevant piece of information has appeared or has been changed, but also effective recommendation how to deal with these changes.

Our approach for managing attention in a business environment is not just to support receiving new relevant information proactively, but also to enable relevant reaction on this information (i.e. on a change in general). Such an action can be an already predefined workflow, but also ad-hock generated procedures according to the currently available knowledge/experience. In that sense, our approach of an enterprise Attention Management System (AMS) goes beyond informing a user proactively that something relevant has been changed, toward proactive preparing and supporting the user to react on that change. Our approach puts forward a comprehensive reasoning framework that can trigger a knowledge base in order to find out the best way to react on a change. We base such a framework on a combination of ECA (event-condition-action) rules and ontologies.

The paper is organized as follows: in the second section we analyze motivating examples, we derive requirements and we
outline a framework for an enterprise attention management system, in the third section we present the architecture of the SAKE attention management system encompassing various functionalities to address relevant attention-related issues and in the fourth section we present existing technologies and tools aiming at supporting management of attention. We conclude by suggesting outlets for future research and development in information technology for the purpose of managing users’ attention in knowledge-intensive environments.

2. REQUIREMENTS AND FRAMEWORK
We define three basic requirements for an Enterprise Attention Management system:
1. Flexible modelling of information in order to enable focusing of attention on different abstraction levels. For example, a user interested in information about pets should be alerted for new information about domestic animals (scenario 1). Another issue is modelling the usage of information by a community of users in order to stimulate sharing of implicit knowledge. For example, users looking for front tyre pressure must be proactively fed with the rear wheels’ pressure as most technicians are interested in both in most maintenance situations (scenario 2).
2. Context-awareness in order to support a particular decision making process. For example, new law about animals triggers different alerts in the GBR and in another business process (scenario 1). Another example is that a new and approved service bulletin should be immediately sent from the airplane manufacturer to all known customers. Moreover, a certain part once rejected from the maintenance list of a service unit should be rejected from all databases (scenario 2).
3. Management of preferences for enabling efficient extraction of interesting information. In particular, there is a need for an expressive attention model for focusing of attention on the given user’s context. In other words, the changeability of resources is proactively broadcasted to the users who can be interested in them, in order to keep them up to date with new information. Users may have different preferences about both the means they want to be notified and also about the relevance about certain types of information in certain contexts. User preferences can be defined with formal rules or more informally by means e.g. of adding keywords to user profiles. Moreover, even when employing mechanisms capable of formalizing the users’ preferences, a certain level of uncertainty about users’ preferences will always remain. For this reason, dealing with uncertainty is an important aspect of attention management systems. Equally important is the way preferences can be derived: by explicitly specifying them or by learning techniques.

3. ATTENTION MODEL
This section presents an attention model which we have developed in SAKE project. The system has been deployed and is currently in the evaluation phase. Unlike other similar models, our attention model proactively supports the user in reacting on changes respecting the user’s context and preferences.

The approach is realized by combining context-aware reactive rules, with ontologies, and an appropriate preference model. Such an expressive attention model is the basis of the SAKE Event-and-Context driven Attention Management System (ECAMS).

Figure 1 presents the conceptual attention model behind SAKE. The model assumes that the interactions between users and internal information sources are logged including the business

non-sensory based, mechanisms also need to be employed to form a complete picture of the user’s attentional state [13].
context (e.g., business process) in which the interactions happened.

Some log entries can be defined as events that cause alerts, which are related to a user, a problem domain and associated to a priority level. Every alert invokes actions, that can be purely informative (i.e., an information push) or executable (i.e. to execute a business process).

Figure 1: Conceptual Attention Model

In the core of the SAKE approach are ECA (Event – Condition – Action) rules; their general form is:

\[
\text{ON event AND additional knowledge, IF condition THEN DO something.}
\]

Relevant events and actions are usually triggered by interactions taking place in organisational systems, such as the SAKE Content Management System (CMS) and the GroupWare System (GWS) or by external change detectors. The later are implemented with the Change Notification System (CNS), a component that can be configured to monitor web pages, RSS feeds and files stored in file servers. The SAKE content management system (CMS) enables storage and provision of content by:

(i) supporting the annotation of content with metadata as well as relations between different content items;

(ii) semi-automatic population of metadata using text mining methods; and

(iii) realizing semantics-based search that retrieves content based on both full-text and metadata.

The SAKE groupware system (GWS) supports information sharing and creation by:

(i) supporting the annotation of the interactions between users;

(ii) enabling identification of communities of practice from mining their interactions and their specific vocabularies by social tagging; and

(iii) searching for experts based on their profiles as these are created explicitly and implicitly during their interaction with the system.

The SAKE Change Notification System (CNS) is a server based change detection and notification system that monitors changes in the environment which is external to SAKE. It can be configured to monitor web pages, RSS feeds and contents of file servers. Users and the administrator can create new notification queries for finding and displaying interesting changes. When creating a query, users can define if they want to monitor a web page for any change or specific changes specified by regular expressions (e.g. new web content containing the keyword “sanitary” but not “pets”).

4. THE SAKE EAMS

Figure 3 depicts overall architecture of the SAKE EAMS system that suits to the general EAMS framework given in Figure 1. The figure also shows other core components of SAKE, i.e. the SAKE Workflow Management System (WfM), Content Management System (CMS), the GroupWare System (GWS) and the Change Notification System (CNS). In the following, we first outline roles of SAKE components which do not belong to SAKE ECAMS, while a more detailed description of ECAMS is given through next subsections.
source crawler, which is used for fetching and parsing HTML documents, RSS feeds as well as resources in other supported formats.

5. SAKE ONTOLOGIES

Conceptual information model in SAKE is realized via ontologies. For example, ontologies are used to model change events, describe various information resources, express user’s contexts and preference rules etc. In the following, we further discuss roles of SAKE ontologies and outline their content.

5.1 Preference Ontology

SAKE proactively delivers information resources (e.g., different documents and files) to a user. The resource delivery is realized in a process of matching the business context on one side, and user’s preference rules on the other side (preference rules are described later in the text). Relationship between the business context and user defined preferences is handled via the validIn relation in the preference ontology, Figure 4 (e.g., particular preference rule is validIn a certain context). The preference ontology is typically imported by another ontology which maps its own concepts to this ontology. More specifically, by subclassing PreferredResource the importing ontology defines for which type of resources (i.e., individuals) the user can define preferences. Similarly, subclasses of ContextObject should be defined in order to indicate which type of individuals the Context consists of (i.e., the business process, activity, task, and the user).

Figure 4: Preference Ontology, Class diagram

Furthermore, we differentiate between a RuntimeContext and a PersistedContext. The RuntimeContext reflects the user’s current context and changes dynamically with the user’s interactions within the system. This context may be used to track user’s behaviour in the system. However, if a user’s interaction is logged in the system as a persistent activity (e.g., the creation of a new document) the user’s current context will be persisted (using the PersistedContext).

The RuntimeContext and PersistedContext are utilised by the Context Observer to extract the current business context, and hence, enable resource delivery based on that business context.

5.2 Information Ontology

The Information ontology (Fig. 5) contains the concepts and relations about information resources for which we want to express preferences, such as documents and forum messages. On the top level we have separated the domain concepts from value types. The FiletypeValue class defines the different file types a file in the SAKE system can have.

Figure 5: Information Ontology - Class hierarchy

In the InformationSource sub-tree we differentiate between information sources which are of an abstract nature (such as persons), external information sources such as Web pages and RSS feeds, and information sources which physically exists in the SAKE system, such as documents, forums or e-mails. We further divided the physical information sources into CMS-specific and GWS-specific entities.

This FiletypeValue class represents the file type (indicated by the file extension) of a document, for example PPT, PDF, DOC, etc. Note that one subclass of filetype can describe multiple file extensions, such as JPG can be a .jpg or .jpeg file.

5.3 Log Ontology

There are many sources of changes that can affect an information resource, like adding, removing, deleting a new document or starting a new discussion. The Log ontology is used for representing these changes in a suitable format. There are four subclasses of Event: AddEvent, RemoveEvent, ChangeEvent and AccessEvent (Fig. 6).

AddEvent is responsible for the creation of new events, e.g. a new document has been added to the SAKE CMS. It contains two subclasses: AddToCMSEvent, meaning the addition of a resource to the CMS and AddToParentEvent, meaning the addition of an existing child to a parent element, e.g. posting a new message to a discussion thread (Fig. 7).
RemoveEvent is dedicated to the deletion of the existing elements from the system, like the deletion of a document from CMS. It consists of RemoveFromCMSEvent, meaning the removal of a resource from the CMS and RemoveFromParentEvent, meaning the removal of a child from a parent element, but the child is still existent.

ChangeEvent is responsible for the modification of an existing individual, e.g., the change in the name of the author of a document. It consists of: PropertyChangeEvent, meaning that some properties of an individual have changed and IndirectChangeEvent, meaning a change caused by some other event.

AccessEvent is dedicated to the access of an existing individual. It represents a very broad class of events like reading a document, for which is very complicated to define the semantics clearly. For example, did someone who opened the document and closed it after five minutes, read the document or just opened, considered it as not interesting, but forgot to close it immediately?

We differentiate subclasses AddEvent and RemoveEvent by addition/removal of resources to/from the CMS and by addition/removal of a resource to/from a parent/child relationship using the isPartOf property. AddToCMSEvent is further differentiated by either creating a resource within the SAKE system or uploading an existing resource. For ChangeEvents, we distinguish between changes of the resource's properties (e.g. metadata) and changes which are caused by some other event.

Properties of an event are the resources the event relates to, the user who originated the event, a timestamp when the event occurred, an optional description of the event and a copy of the current runtime context. In the case of ChangeEvents we add the names of the resource's changed properties, and optionally link to another event which caused this ChangeEvent.

The following considerations are assumed:

- An information resource is associated with multiple events, the first event is always an AddToCMSEvent (addition to the CMS). Then multiple events of different kinds can follow. After a RemoveFromCMSEvent (removal of the resource) no other events follow.

- In order to see which aspect of the resource has changed, we associate the names of the changed properties with the event.

- If the event has been caused by another event (e.g. the modification of a thread has been caused by the addition of a new message, see below), then an IndirectChangeEvent is generated which links to the original event via the causedBy property.

Special attention has to be paid if we define events for "compound" resources, i.e. resources which have child resources. Consider a forum in the SAKE GWS for instance: There we have forums, which consist of multiple threads which consist of multiple messages. Now, imagine that someone adds a new message to a thread. It is clear that by adding a new message to a thread we can consider the thread as changed, thus creating an AddToParentEvent for the message and an IndirectChangeEvent for the thread.

In order to resolve this issue, we define the following as the default behaviour:

- Compound resources state their parent/child relationship by the property isPartOf or a sub-property thereof.

- For compound resources, a ChangeEvent will be generated if (i) the properties of the resource itself change (i.e. title of a forum thread changes) \( \Rightarrow \) PropertyChangeEvent; or if a child object has been added or removed (e.g. adding a new message to an existing thread) \( \Rightarrow \) IndirectChangeEvent.

- The modification of a child object does not result in a modification (i.e. IndirectChangeEvent) of the parent.

- The developers programmatically create only the most basic event, e.g. a PropertyChangeEvent or a AddToParentEvent. SWRL rules decide whether this event triggers an IndirectChangeEvent or not as described in the following paragraph.
IndirectChangeEvents are never created by the developers.

We do not hard-code the propagation of events from child to parent, instead we define them in SWRL rules (cf. Figure 8).

\[
\text{addToParentEvent}(?, E) \land \text{resource}(?, \text{RES}) \land \text{hasEvent}(?, \text{RES}, ?) \land \text{relatedParent}(?, \text{RES}, ?, \text{RES2}) \land \text{swrlx}:\text{createIndividual}(?, E2) \Rightarrow \text{indirectChangeEvent}(?, E2) \land \text{hasEvent}(?, \text{RES2}, ?, E2)
\]

**Figure 3:** An example of the SWRL rule

Default rules state that the addition/removal of a child object triggers a ChangeEvent for the parent object. However, in order to be more flexible, we could also state that the modification of a specific child object also causes the modification of its parent. Note that in this way, we may use events to specify more complex events (e.g., indirectChangeEvent). Those complex events are created using SWRL\(^2\) (Semantic Web Rule Language) rules and a number of built-in predicates supported by KAON2\(^3\). Although realised in a declarative way, Complex Event Processing (CEP) in SAKE is still limited, and it is subject of our future work. Particularly, we will continue developing declarative CEP. The advantage of such an approach is that definition of a complex event may easily be altered by changing only a logical rule. Further on, inconsistencies in CEP are handled by means of logic.

6. PREFERENCE MANAGEMENT

There are two basic tools responsible for managing preferences: Preference editor and Reasoner. In this section we give more details.

6.1 Preference Editor

A preference is an n-ary relation between a user, multiple resources, and a preference value. Figure 9 shows how such a preference relation is formally modeled using the preference ontology.

![Figure 9: Preference as n-ary relation between a resource, value, user and rule](image)

Each preference (i.e., n-ary relation) is expressed as a logical rule, represented in SWRL. Figure 10 illustrates a preference rule: if userA is in the processZ, then userA has preference of value 1.0 for documents created in 2006. Among the preferred values, preferences include the business context of the user, in order to support context-awareness of the whole system (e.g., userA and processZ are related to each other by the same runtime context: ctx).

Utilising logical rules, for expressing context-sensitive user preferences, SAKE features a very flexible preference model. One rule is used to assign different preference values to different information resources based on relevant criteria of a particular user. Therefore every information resource may be assigned with different preference values by different preference rules (i.e., by different users and/or business contexts). Another flexibility of the SAKE preference model comes from an implicit representation of preferences. Since preference values are not pre-computed and persisted in the system, just adding one preference rule may significantly influence the whole preference model. Also adding a common preference to the SAKE preference model (i.e., a preference valid for all users) may be as easy as adding only one preference rule. Moreover updating existing resources, or adding new ones, does not mess up all previously created preference values. In this way, a user is given a great freedom to create particular preferences for particular processes, activities, tasks, and even to aggregate multiple preference values\(^4\) for one resource into a final score.

![Figure 10: A Sample Preference Rule Expressed in SWRL](image)

The Preference Editor supports creation of preference rules by providing a GUI for step-wise, interactive rule development, as presented in Figure 11. The user starts with selecting what kind of resources (i.e., file, forum, workspace, email etc. that is a subclass of pref:PreferredResource) s/he wants to define a preference for. This information is specified in the information ontology (Fig. 4), and is represented as a variable ?RES in Figure 10. The preference rule, is than, further extended narrowing down the preference criteria in several subsequent steps (possible introducing new variables). For each of these steps, SAKE reasoner (described in Section 6.2) is used to find out the list of possible properties or property values that are available. Further on, values entered by a user are syntactically checked out (e.g., for the data type). In this way, the Preference Editor eases the process of creating valid and consistent preferences.

![Figure 11: Preference Editor: Step-wise, interactive rule development](image)

Preference rules, created by the editor, are serialised to its SWRL representation and stored in the preference ontology. Finally, preference rules may also be removed (or updated) using the Preference Editor.

---

2 http://www.w3.org/Submission/SWRL/

3 http://kaon2.semanticweb.org/

4 Preference aggregation is computed in cases where there exist more than one preference rule for an information resource.
6.2 Reasoner

Rule-based, formal, reasoning plays a central role for managing user’s attention in Sake. The knowledge base of the system comprises of nine Sake specific ontologies. The purpose of these ontologies is to formally describe information resources, preferences, and events in Sake, supporting a user in knowledge-intensive and highly dynamic business environments.

As we already explained in the previous section, a preference is expressed as a logical rule, represented in SWRL. The rule matches an information resource with the desired criteria (i.e., particular business context etc.) and assigns a preference value to them. Preference rules are stored in the Sake knowledge base, and they are evaluated by the reasoner when certain events occur within the system. Events are triggered whenever a user changes her/his run-time context. A user may utilise these events to create Event-Condition-Action (ECA) rules. ECA rules offer extensible and flexible approach to realizing active Knowledge Management Systems. Such systems are enabled to actively respond on events or state changes.

Currently we allow user to use preference rules as the action part of an ECA rule, and do not explicitly define the condition part. Since a few preference rules may be defined for one particular event, Sake may be seen as a reactive system capable of executing broad form of complex actions (i.e., actions that comprise of simple or atomic actions). Enhancing Sake with more complex operators, for combining atomic action into complex actions, is a subject of our future work.

It is a task of the reasoner to react on events and evaluate appropriate ECA rules. The evaluation procedure is as follows: on an event, the system issues a SPARQL query which starts the reasoner engine. The reasoner takes into account a relevant preference rule (with the respect to the SPARQL query) and evaluates the rule. In case, there are more than one preference rule, defined for a particular information resource, the reasoner applies an aggregation function combining the single preference values into an aggregated score.

Once the rules are positively evaluated, a notification is generated and provided to the user both as a web bulletin and an RSS feed. The RSS feed points to a web page containing the results of the evaluated rules. By adopting RSS as a notification mechanism instead of sending emails, we allow the user to select whether s/he wants to be notified immediately about relevant information, periodically or when s/he wants to be notified. Most RSS readers allow the user to specify when and how they want to be notified about updates in the feeds they receive.

We use KAON2 as an underlying inference engine for managing, querying and reasoning about the ontologies in Sake. KAON2 supports the SHIQ(D) subset of OWL-DL, and DL-safe subset of SWRL rules. Also a good portion of (but not entire) SPARQL specification is supported, and a number of built-in predicates have been implemented in this reasoner.

7. IMPLEMENTATION

The Sake prototype is based on J2EE and Java Portlets following a three-tiered architecture (Fig. 11). The presentation tier contains Portlets, JavaServer Pages (JSPs) and an auxiliary Servlet. Portlets call business methods on the Enterprise Java Beans (EJBs), pre-process the results and pass them to the JSP pages. The JSPs contain Hypertext Markup Language (HTML) fragments as well as placeholders for dynamic content (such as the results passed from the Portlets). The auxiliary Servlet is used for initializing the connection to the KAON2 ontology management system (http://kaon2.semanticweb.org/, part of the integration tier).

The business tier consists mostly of EJBs, which provide the business logic and communicate with the components of the integration tier that comprise a third-party CMS component (Daisy) and GWS component (Coefficient) as well the Preference Framework. The interface to these components is represented using EJBs which all use the Kaon2DAO in order to access the ontologies: the CMSBean and GWSBean enhance the CMS and GWS with semantic meta-data, the AMSBean manages the preference rules. KAON2 stores the semantic meta-data for these entities with ontologies and provides the facilities for querying them using SPARQL. The KAON2 reasoner is used for evaluating the user’s preference rules. The integration tier contains also a MySQL relational database, which stores CMS- and GWS-related content, such as forums, discussions, documents etc.

Figure 12: Sake Technical Implementation

Since the development of the Sake system has not been completed yet (mainly integration of components is still pending), a comprehensive user-driven evaluation of the system as a whole is planned but not performed yet. On the contrary, we have performed an early evaluation of the main Sake components, independently. Evaluation has been performed in three case studies: two local administrations and one ministry. We validated the usability of these components and their relevance to the knowledge-intensive work of public servants. We collected useful comments for further improvement regarding the functionality and interface of the Sake components. Early evaluation of the Preference Framework in particular has revealed noticeable improvement in relevance of system-generated notifications when user preferences are taken into account. In the future we plan to perform formal experiments to measure the degree of improvement of notifications. Moreover, as soon as the Sake system is integrated, we plan to test the system’s ability to not only send relevant notifications to users but also execute relevant actions such as the initiation of a workflow process.

From a conceptual point of view, we have ensured that all components are based on a common ontological model for representing information resources and changes as well as other...
8. CONCLUSION
In this chapter we presented a novel approach for managing attention in an enterprise context by realising the idea of having a reactive system that manages not only alerting a user that something has been changed, but also supporting the user to react properly on that change. In a nutshell, the corresponding system is an ontology-based platform that logs changes in internal and external information sources, observes user context and evaluates user attentional preferences represented in the form of ECA rules. Since the presented system is currently under deployment in a real-environment, results from the formal evaluation are still missing. However, initial assessments have shown that the users find the system useful and usable. Additionally, efforts to setting-up the whole system are limited, which is one of very important characteristics of semantic-based application.

Future work will be toward further refinement of ECA rules for preference description and automatic learning of preferences from usage data using and machine learning techniques. Considering the existence of hierarchical relations that exist in the Log Ontology which models interactions (events) that happen in the SAKE system, we can further utilize Generalized Association Rules in order to mine and discover interesting patterns in the system’s usage. Generalized Association Rules improve upon standard association rules taking into account a taxonomy. In particular, let \( I = \{i_1, i_2, \ldots, i_m\} \) be a set of items. A GAR is a deduction of the form:

\[
X \Rightarrow Y, \text{ where } X, Y \subset I, \text{ and } X \cap I = \emptyset \text{ and no item in } Y \text{ is an ancestor of any item in } X.
\]

For example, GARs can help discovering that many users involved in business process X are referring to documents created by employees of department Y or that they often consult the same employees by exchanging forum messages.

Additionally, future work will tackle the problem of the automatic definition of events. Indeed, in the current version events are defined in advance and “hard-coded” in the process workflow, which supports execution of events, but decreases the adaptivity of the system. We will work on methods for event discovery from log data, which will enable dynamic changes in the configuration of the system.

Acknowledgements. Our thanks to the European Commission for partially funding the SAKE project.

References


