AlertMe: A Semantics-based Context-Aware Notification System

Asterios Leonidis\textsuperscript{1,2}, George Baryannis\textsuperscript{1,2}, Xenofon Fafoutis\textsuperscript{1}, Maria Korozi\textsuperscript{1,2}, Niki Gazoni\textsuperscript{1}, Michail Dimitriou\textsuperscript{1}, Maria Koutsogiannaki\textsuperscript{1}, Aikaterini Boutsika\textsuperscript{1,2}, Myron Papadakis\textsuperscript{1,2}, Haridimos Papagiannakis\textsuperscript{1}, George Tesseris\textsuperscript{1}, Emmanouil Voskakis\textsuperscript{1}, Antonis Bikakis\textsuperscript{2} and Grigoris Antoniou\textsuperscript{2}

\textsuperscript{1}Computer Science Department, University of Crete, Greece
\textsuperscript{2}Institute of Computer Science, Foundation for Research and Technology Hellas, Greece

\texttt{leonidis@ics.forth.gr, gmparg@ics.forth.gr, fontas@csd.uoc.gr, korozi@ics.forth.gr, ngazoni@csd.uoc.gr, mdimitr@csd.uoc.gr, mkoutsog@csd.uoc.gr, boutsika@csd.uoc.gr, myrpap@csd.uoc.gr, xapapag@csd.uoc.gr, tesseris@csd.uoc.gr, voskakis@csd.uoc.gr, bikakis@ics.forth.gr, antoniou@ics.forth.gr}

Abstract

In this work we present “AlertMe”, a semantics-based, context-aware notification system that provides personalized alerts to graduate students based on their preferences. An extensive description of the system is carried out. We present the underlying ontology that models the available knowledge, as well as how higher level knowledge inference and context-based decision making is achieved through rule-based reasoning. Finally, we outline the technical aspects of the developed system, covering issues involving the integration of the various subcomponents.

1. Introduction

In recent years, application designers are continuously searching for ways to offer personalized services of a higher level to an ever-growing user base with a multitude of needs and demands. These services exploit information from many different aspects of a user’s behavior such as her location, whether she is a member of a working group, her personal likes and dislikes and so on. All this information falls under the notion of user’s context. Context allows information to follow the user and not vice-versa, as it has usually been the case until now.

Context-awareness and personalization are crucial in the design of personal notification systems and alert services, because they contribute to the satisfaction of needs such as productivity improvement, timely, reliable and instant updates, effective cooperation support and workgroup coordination. Such needs have been constantly parts of well-organized and competent academic communities. “AlertMe” attempts to address such needs by offering personalized alert services to graduate students.

Graduate students are open to multiple sources of information as members of the academic community of their university. In many cases, it is desirable, if not demanded, that such information reaches the right interested party at the right time, or before a preset deadline. We have identified three distinct types of information that interested parties might want to access through personalized alerts, and thus being at the focus of the system we developed.

The first category of alerts are course-related and aim to inform students about canceled lectures, changes in the timetable or the rooms used for the lectures, extra lectures and other related events. These alerts should allow students to keep up with the current schedule of their attending courses each semester.

University or campus-related alerts constitute a separate category of notifications that handles information, which refers to the academia in general. Alerts in this category can derive from the graduate students’ secretariat, the library and other campus entities. Finally, a third source of alerts is related to groups in the academic community that either share the same interests (e.g. photography, sailing or painting groups) or work in the same project, participating in the same workgroup or working team.

AlertMe exploits the advantages of Semantic Web technologies in order to provide the alert services described above. Ontology languages assist in creating a knowledge model that is rich and expressive, as well as reusable and extensible. Moreover, rule languages enable reasoning about the user’s context, needs and preferences, which are encoded in the ontology model.

The rest of the paper is organized as follows. Section 2 discusses related work on personalized notification systems. Section 3 presents a motivating scenario from the University domain. Section 4 presents the knowledge representation model in the
form of an OWL ontology. Section 5 describes how rules are applied on the available knowledge to enable context-based reasoning. Section 6 offers a thorough description of the final system, its user interface and the services it provides, and Section 7 concludes.

2. Related Work

Campuses and academic communities have been the targets of many existing context-aware systems and applications that appear in literature. In [1], the authors present the work of the MyCampus group at the Carnegie Mellon University, which includes offering context-aware services to campus users. These services offer among others recommendations based on user preferences and user location, task reminders, crime alerts, and support for collaborative applications.

eyeJOT [2] is another context-aware campus information system that supports information posting for news, activities and schedules using Short Message Service (SMS). This work combines mobile devices with wall-sized ambient displays, and uses proximity detection as an interaction mechanism in order to inform campus users either by showing information directly through the displays or by sending messages to the users’ mobile devices.

e2Campus [3] is a commercial emergency notification system that empowers dispatching of time sensitive messages, such as emergency broadcasts, school closings and All-Points-Bulletin. The key feature of e2Campus is that it offers a centralized platform to deliver notification to an entire audience by all means, including SMS messages.

Apart from university campus-specific applications, mobile devices have been exploited to offer immediate notification services to end-users in general. The Hong Kong Institute of Education Library [4] has developed an alert service to further enhance their communication with library users. This is achieved by integrating the mobile phone SMS and Google Calendar technologies to provide simplified versions of important library notices, such as availability of requested items, overdue reminders and so on.

Finally, IBM offers, through its WebSphere suite a set of Intelligent Notification Services [5] based on user preferences and subscriptions. Intelligent Notification Services supports the following types of notifications: direct notifications sent by users or other applications and subscription-based notifications. Most popular delivery channels are supported including SMS channels for sending short text messages to mobile devices.

Our approach, unlike these systems -with the exception of MyCampus- takes advantage of Semantic Web technologies such as ontology representations and rule-based reasoning to provide personalized services, and boosts performance by employing caching techniques. Unlike MyCampus, AlertMe focuses on delivering live, direct notifications by employing SMS services.

3. A Motivating Scenario

In order to illustrate the needs that motivate the development of “AlertMe”, consider the following scenario. A graduate student, Kate, decides to attend course CS566 this semester. A lecture is scheduled for today. However, due to an unforeseen event, Prof. Linus has to cancel the lecture, so he needs to notify the attendees immediately about the cancellation and reschedule the lecture.

Kate attends the rescheduled lecture. She is appointed as the head of a project team and sets a date for the first team meeting. When the day of the meeting comes, she needs to contact the team members and inform them about the meeting location. She also wants to be reminded that, while she is at the campus premises and before the team meeting, she has to visit the library to collect material related to the team project. Kate is also a member of the campus photography group, which wants to inform its members of an exhibition that is worth checking out.

The final date for the course subscriptions for this semester is approaching and the secretariat needs to inform all students to submit their applications before that date. The secretariat decides to inform the students every other day until the final date, so that no student misses the deadline.

These are a few cases where a context-aware notification system such as “AlertMe” can be used. This scenario highlights the multiple sources of notification in a typical academic community, and the facts that each notification is addressed only to a subset of that community, that some notifications are more urgent than others, and that some others need to be repeatedly sent to their recipients. Finally, it should also be noted that, given the multitude of alerts one student can receive, depending on her degree of involvement in campus activities, it would be preferable to group less urgent notifications together, as a single notification.

4. Modeling knowledge

For the purpose of modeling and representing the knowledge that is inherent to our system, we created an ontology schema that we present in this section. OWL [6] was our choice for modeling language, mainly due
to the fact that it enables reasoning with SWRL [7] rules for the decision making process, as described in this section. Although we do not fully exploit the expressivity of OWL, since the modeling requirements are not very demanding, our model can be easily extended with more complex OWL constructs that enable imposing several types of constraints on the classes and properties of the schema, in case we want to add services that demand more expressive modeling features.

The ontology that represents the knowledge for our system consists of six main classes which are interrelated through several relations. These classes and the relations interconnecting them are depicted in Figure 1. In the rest of this section, we will briefly describe each one of the classes, explaining what type of knowledge is captured by each one. We also describe the most important properties of the classes.

4.1. Modeling people and courses

As illustrated in Figure 1, the main class is Person. This class models any kind of person, either a faculty member or a student, while a number of properties describe special characteristics for each person. Students and faculty members are modeled as two distinct subclasses of the Person class, which represents common interest groups and workgroups, through two different relations. Property belongsTo represents the fact that a person belongs to a team, while legitSenderFor denotes that the particular person has the privilege of sending alerts to all team members.

![Figure 1: Ontology Schema](image)

The system captures the knowledge about the courses offered to graduate students through the Course class. The need for this class is dictated by the fact that alerts can be course-specific, having as destination the attendees of a course. To that effect, instances of Course are linked with a property to instances of the Student class, to denote that a course is attended by a particular student. A separate link exists from Course to FacultyMember, through the property teaches, which allows denoting the teacher of each course.

4.2. Modeling alerts

Since the developed system is an alert-notification system, the core class in the underlying ontology is the Alert class. This class represents a placeholder for all the aspects of an alert not only in the real world but also in the system. Each instance of this class represents a single alert with multiple possible destinations.

Alerts have three special characteristics: their ability to be grouped, their urgency level and their repeatability. A number of alerts can be grouped together and sent through a single message. The property isNongroupable is used to denote whether an alert can be grouped or not. By default, all alerts can be grouped. However, in order to avoid long delays in sending alerts while waiting to group them together, a special property (repeats) that keeps track of the times an alert has been put on hold is introduced. Furthermore, alerts can be flagged as urgent (using the isUrgent property) when it is important to inform the users immediately, without waiting for any other alerts to group with.

As far as repeatability is concerned, alerts can be sent repeatedly in predefined periods, to act as constant reminders for a particular event or deadline. A threshold for the maximum number of times an alert can be sent is set by the sender, as well as a period of time during which an alert is active. These two different ways of limiting the repeatable alerts are necessary since sending large number of alerts can be not only costly but also annoying to the users.

4.3. Modeling teams and categories

To model different types of groups that may be formed in a campus based on similar interests or cooperation in projects, the Team class is used. A team consists of Person instances. For example, a person can belong to a team related to a course or to a project, or to other activity groups like sailing or photography groups. Every team has a distinct topic. Topics are modeled by a separate class, called Topic, which contains information describing the interests and purposes of the team.

Finally, the Category class is used for modeling various categories of alerts and is the range of many relations in the ontology schema. Each Category
instance models a type of sender. For example, there are alerts that are being sent from the graduate students’ secretary, the sailing team, and so on. For each one of the distinct sender types, there is a Category class instance that models them. To that end, each Alert instance is linked to a Category instance that defines its sender. Moreover, each Team and Course instance is also linked to a specific category, since they are both sources of alerts. All these links are realized by the hasCategory property. Student instances are also linked to Category instances through hasInterestIn property, to denote that a specific student is interested in receiving alerts that belong to those categories.

### 5. Decision Making

The intelligence of a context aware system depends to a large extent to its ability to infer new knowledge from context information. The “AlertMe” system takes advantage of semantic technologies in order to make context-dependent and policy based decisions. The available context information is modeled through the ontology described in the previous section. The final decision is drawn by the Reasoner, a rule engine, based on policies expressed as a set of SWRL rules.

Semantic Web Rule Language is a proposed prototype developed in order to express such rules, providing interoperability between different systems. At present, SWRL aims to be the standard rule language for the Semantic Web. By adopting it, our system inherits all the benefits of using a standard instead of a specific implementation. The rules do not limit the programmer to using a specific platform, programming language or even operating system. As a result, “AlertMe” is not limited to a specific rule engine since any SWRL-compatible rule engine can be applied. Generally, using SWRL has additional benefits. Firstly, changes can be made in the ontology without the need to recompile the implementation code. Moreover, the fact that the ontology and rules are not bound to the implementation provides a more easily extensible platform.

For the implementation of the “Alert Me” decision making module, the Protege-OWL [8] libraries are used for the purposes of ontology management and Jess Rule Engine [9] is used for the purposes of rule handling. Additionally, the swrl-jess-bridge library, part of the Protege libraries, is responsible for bridging the ontology to the rule engine.

#### 5.1. Rules

The system uses the Reasoner in order to determine how to handle alerts, users and workgroups. In particular, a set of rules defines which users will be the recipients of each alert. Another set defines the conditions under which alerts can be grouped and sent in the same text message. Other rules denote whether a user has the privileges required to send alerts to a specific team, when an alert will be dropped, or to automatically populate a user’s profile with her interests. For the purposes of these operations, a set of SWRL rules is defined.

Due to space limitations, we will describe only two of the most important rules. The first rule describes the conditions under which a new alert should be sent to a user.

\[
\text{Alert(?a) ^ Student(?u) ^ isEnabled(?u, true) ^}
\]

\[
\text{hasInterestIn(?u, ?cat) ^ hasCategory(?a, ?cat) ^}
\]

\[
\text{repeats(?a, "0") →}
\]

\[
\text{receives(?u, ?a) }
\]

To receive a certain alert, the user should have already activated the notification service and his interests should include the category of that specific alert. Note that this rule supports only new alerts, whose current iteration field is zero. For repeatable alerts, additional rules that check user’s desire to receive alerts multiple times are applied.

The second rule handles the repeatable alerts and decides whether the alert should be resent or has expired.

\[
\text{Alert(?a) ^ repeats(?a, ?r) ^ maxrepeats(?a, ?r) →}
\]

\[
\text{hasExpired(?a, true) }
\]

The author of each alert defines the number of iterations, while an alert is considered expired when it has reached the maximum number of its repetitions. Note that the same rule could also be applied to non-repeatable alerts when their maximum iterations’ field is set to “1”.

#### 5.2. Performance Improvements

The performance degradation was the price to pay for using SWRL rules. Previous work on similar platforms indicates that decision making using Ontologies and SWRL is extremely expensive in terms of time consumption [10]; therefore not being suitable for time-critical operations. However, the latter is not a fundamental problem of SWRL but rather a problem of the Protégé implementation. Due to this fact, simplicity was the primary concern when designing the decision making module.

The system does not use the ontology for storing user, team and alert profiles; an ER database is used instead. The ontology is solely used to support the decision making process. Upon every decision request,
the necessary data are imported to the ontology, exploited for decision making and then removed to preserve ontology's explicitness and decision making process efficiency.

Figure 2: Moving average of the duration of the decision making process

Moreover, a caching system was integrated to the decision making module to save computational power (Fig. 2). The latter is achieved by caching specific decision to specific inputs. For the implementation of our caching system, the Apache’s Java Caching System [11] was used. Depending on the nature of each rule, caching decisions can be very efficient. When the set of distinct input values of a rule is finite, caching can yield very high hit ratio, after a sufficient number of iterations; assuming sufficient memory.

For instance, integrating caching to the decision regarding the expired alerts (described in 5.1), yields significant performance improvements. Figure 2 depicts the rolling average of the duration of the last 20 decisions on uniformly random alerts after the total number of iterations (x-axis). For the purposes of the experiment we assume that the maximum iteration field (maxrepeats) is uniformly distributed on [1, 10] and the current iteration field is uniformly distributed on [0, maxrepeats]. Note that there are only 65 distinct inputs. This indicates the low memory requirements.

Concluding, it is important to note that “AlertMe” does not contain any time critical operations. Thus, adopting this approach did not reflect badly in the provided services.

6. Integration

With respect to the implementation aspects of the “AlertMe” system, the next sections will present in more detail our approach from a technical point of view, along with the addressed integration issues.

6.1. “AlertMe” Bean Approach

“AlertMe” is information oriented as information is used both in decision making and in the context of briefing or updating users. Thus, there was an imperative need to identify the different information units and design the appropriate classes. These units are information carriers and provide functions that facilitate retrieval or modification of the contained data. As a result, they are independent from the alternative data sources, and from their possible users.

Java beans [12] software pattern was a perfect match for our criteria. Java Beans are classes written in Java, and conforming to a particular convention. Their main usage is to encapsulate many objects into a single object (bean), so that they can be passed around as a single bean instead of multiple individual objects.

6.2. Database Management

The Database Management Component (DBC) simplifies the communication between the “AlertMe” system and the underlying database system. The key feature of this mechanism is the provision of a unique entry point to the database, where the respective component handles all the database-related actions and hides any transaction complexity. The DBC acts as a transparent proxy that establishes connections with the database and facilitates data insertion and extraction in a uniform manner, being totally unaware of any “AlertMe” bean semantics.

6.3. “AlertMe” Data Holders

As mentioned above the Java Bean technology [12] was used to eliminate the data exchange complexity. Though, data retrieval from the relational database is handled by special components, called holders. These components are aware of the DBC and able to perform queries in order to populate their contained collections of beans, while “AlertMe” utilizes holders to manipulate data efficiently. The key feature of holders is the sophisticated selection of the appropriate collection types to boost performance in the most common use cases.

Each holder is aware of the retrieval and storage queries regarding its hosting bean type. Subsequently, when information loading is necessary, the appropriate holder queries the database, handles data marshaling and updates its internal collection, thus making the loaded information publicly available. Respectively when information storage is necessary, the appropriate holders are responsible for handling that needs.
6.4. Scheduler and SMS Platform

“AlertMe” enforces SMS grouping to save credits and support repeating alerts. The Linux CronTab manager was selected to schedule the periodic invocation of the component responsible for gathering and delivering the alerts.

The SMS Delivery mechanism is realized via a “Group SMS” service [13], where the execution stages are: connection initialization, data –message and recipients’ list– transmission, and connection purgation upon successful delivery.

6.5. “AlertMe” Web Interface

“AlertMe”, apart from delivering notifications to end-user mobile phones as SMS messages, also provides a web interface component that facilitates service personalization and calendar population. The composition of “AlertMe” system’s target group of intermediate users—in terms of web expertise— yielded the need for a straight-forward interface. Subsequently, the design was based on the Web Usability approach and the controls were grouped into operation specific categories—such as alert overview or system personalization— to facilitate interaction without requiring any special skills or former training.

7. Conclusions and future work

In this work, we presented a semantics-based, context-aware notification system that provides personalized alerts to graduate students based on their preferences. We analyzed the underlying OWL-based representation model, and described how it captures the available context knowledge. We also outlined the decision making process that is performed by the rule engine of the system. Finally, the technical aspects of the system were illustrated, showing how the system functionality was developed and how the various subcomponents were integrated.

Currently, a prototype version of the system has been developed and tested. In the future, we plan to augment the system functionality as follows. First, instant messaging facilities can be added, possibly by including an address book in the user profile. Second, a Team Management component can be integrated to the current system, in order to provide functionality similar to the one offered by well established social networking communities. Finally, an SMS “billing” mechanism can be developed, so as to prevent malicious users from repeatedly sending useless alerts, effectively eliminating spamming.

References