Semantic Multicast Notifications: Increasing information timeliness and efficiency with ad-hoc mapping of operational events and organizational data

ZAMIR DIKA, VISAR ELMAZI
Faculty of Contemporary Sciences and Technologies
South East European University
Ilindenska St. 1200 Tetovo
MACEDONIA
z.dika@seeu.edu.mk, v.elmazi@seeu.edu.mk

Abstract: – In this paper we analyze the role of notification systems in group collaborations and in the SEE University and then we evaluate student opinions about notifications. We then propose an Ad-Hoc Semantic Multicast Notification System which increases the quality of notification services and provides a better connection between the students and the university through the use of modern messaging technologies. Finally, we compare the findings with our quality research and estimate the potential improvements.

Keywords: – organization, operations, semantics, events, notifications, database, ad-hoc, messaging system

1. Introduction
The higher education “market” is ever increasing and with that the number of universities also. Universities are attempting to get competitive advantage from the quality of administrative services, the timeliness and efficiency of the information and services offered to the students, besides focusing on the quality of “academia”. The SEE University, although a rather “new” university has been put to the test to face with obstacles which even more mature universities from the country and the region are barely capable of. While trying to bridge the gap between traditional studies model and the modern, Bologna oriented approach; the university is trying to move towards more student oriented administrative services in order to position itself firmly in the market. Notifications are crucial in current busy schedules and diverse and scattered operation environments, by which we determine when, what and how certain updates in one system notify other systems or other user interfaces [1]. Contemporary notification systems are used extensively in university campuses for broadcasting alerts to staff and students or unicast notifications about certain specialized events or updates. These systems are very efficient for mass notification but seem to become ineffective when required to target selective users in dynamic environments in the form of multicast messages. Multicast messages are very important in group/team collaborations but are hard to implement without having the users subscribe the feeds first. Maintaining these groups manually, especially if they are frequently updated, requires additional unplanned manpower and expenses and as a result, this is becomes very inefficient. In order to discover which users belong to each context and dynamically generate multicast groups in the time of event and mapping and notifying those groups with the event is a prime challenge in our environment [2].

2. Actual notification system
Currently the university’s faculty and administrative staff are mostly relying on conventional notification methods and as such they have been accustomed to more traditional approach for notifying students, in e.g. professor publishes grades in an announcement board, or at their office doors; administrative staff informs students about cases mainly by being present at their offices, etc. In interim interviews with random students, we have discovered that many full-time students; regardless of them being present in the university campus they use unreliable information sources mainly from their colleagues or other students. The part time students acquire their needed information mainly through electronic services, namely E-Mail, E-Service and E-Learning. In a recent study, we performed Notification Quality and Timeliness poll from the student’s perspective, where participated just over 10% of all the active students. We discovered that 87% percent of all the participants expect their notifications on E-Mail or E-Service and 75% of all of them at least once per day check their university mailboxes or open E-Service to check for news (Figure 1).
This obviously is in contrary to what they are currently getting from most of their professors and from the administrative staff. On the other hand, only 34% of participants consider themselves well informed, which is a relatively poor grade (Figure 2).

Furthermore, 95% of all participants believe that an active notification system would have increased the quality and timeliness of information they receive and thus they would have perceived the whole information system better, in the given terms of measure. This is an important finding; because it discovers to us the particular field of improvement we should focus on and the results we can expect should we choose to follow that path of study. From these findings, our initial conclusion was that we should increase the level of information we provide on E-Service and the frequency and context of information we chose to display there.

When asked what would have been their medium of choice for receiving notifications, the students have been given option to choose among E-Mail, SMS (text), Facebook or Twitter; 72% of those who voted for E-Mail prioritized it as the most important, whereas did 54% for texting (SMS), 33% for Facebook and only 8% for Twitter. Thus we have concluded that besides E-Service, as the primary communication medium with our students, we should also send E-Mail for more important issues and also SMS for the most important notifications. But, since our current communication tools are more or less suffering from the similar weaknesses of conventional notification/messaging systems, besides broadcasting and unicast messages, we must use some kind of dynamic multicasting in order to achieve the level of functionality that we want to achieve.

We came to conclude that we needed a solution which would enable us to dynamically extract group membership from operational data, organization database, course rosters, course enrollments, and other organizational data into lookup tables. This generated data should be periodically updated from various operational databases and indexed for lookup searching. These concepts have slowly developed into a mature idea of a new dynamic semantic notification system which should address all the above issues and weaknesses in a more intelligent method.

3. Semantic notification system

An ad-hoc notification system will be able to intelligently build multicast notification groups depending on the context of the event in e.g. grade has been published, exam session has started, class schedule has changed, course homework has been published, curriculum has been updated, etc. These events, depending on the context carry event metadata, in the form descriptors in e.g. class = CBA001/06-12, teacher = z.dika, timestamp = 2009-09-18 12:50, etc. These descriptors then can be used to perform queries on indexed organizational data with the descriptors as keywords and slowly narrowing the multicast group to couple of people to notify. Addressing this way the problem, allows us to decide faster without requiring fully reliable information and coming to better and faster multicast groups without requiring human intervention.

By building such an ad-hoc system which operates autonomously, we will be able to also feed with events from legacy systems, when possible, and in that way lower financial overheads incurred by removing or upgrading older systems. Furthermore, this kind of system if extended with web services, would virtually allow any other legacy system to directly notify when events happen or if that would not be possible. From the development and maintenance perspective, this system would have liberated the developers from the tedious tasks of creating communication channels with different delivery methods for different applications and operations, but instead they would generate a single notification and deliver it to our system and the system would then autonomously decide the most appropriate people to be notified with the event. One very important aspect is that this approach would also allow direct human intervention in special operational circumstances to notify a specific group of people about specific events semi-autonomously, in e.g. notify the German language seniors that a special scholarship is available for them, or
notify students who have not passed the exam = CCS60 with teacher = v.elmazi that the passing deadline is the give date, etc. This configuration has been reviewed from colleagues and peers in the department of IT services and evaluated and validated as very inexpensive solution both in terms of required time of implementation and technology and infrastructure constraints.

3.1 System organization and description

In essence the new system can be described like a combination of an agent which gathers organizational data across the organization from known data sources and a web service which accepts events from many applications and systems with the corresponding event descriptors. In addition, a mapping component which acts on event generation and queries organization data with event descriptor in order to create a list of people related with the event. And finally, a message queuing system which selects the best method of delivery in e.g. E-Service, E-Mail, and SMS and delivers the message to the selected people related with the event (Figure 3).

![Figure 3. System View: Original semantic multicast notification system idea](image)

This original configuration has been then further implemented into real agents, components and services until we have reached a valid solution. The solution consists of: organizational data gathering agent - which indeed is a daemon service that continuously passes all known databases that contain organizational data and updates an indexed database of name, value pairs and a membership key that creates referential relation of one key, value pair with another. This way we are able to create large trees of hash tables that can refer one another at any point; event queuing service – which is a standard XML web service that authenticates known applications and gives them the option to add events with descriptors; event-organization data mapping component – which is an algorithm which performs fast search through the organization tree to look for key pair values, or by referential membership; and message delivery service – which also is a daemon-service that runs in a separate process and delivers messages (notifications) to the target recipients specified by the mapping component. The delivery methods are currently predefined (E-Service, E-Mail and SMS), but we can use virtually any deliver method, as long as we have a component which interfaces the device or service that performs the actual delivery, in e.g. Facebook Apps, Twitter, Pager, but also public notification services like Voice/Loudspeaker, billboards, displays, etc.

3.2 Organizational data gathering component

Organizational data are scattered across many databases in the organization, especially project organization, custom and dynamic teams. There are two types of organization data that we want to gather: static data and dynamic data. Static data are data about employees and positions and buildings and offices. Dynamic data are data about projects, teams, operations, developments, classes, exams. Since in both types of data the key link is the human factor, we first create the list of all the people in our organization with the required contacts addresses. Because the data structure of the index that collects data organization is a tree structure and within the scope of the project we only want to collect their username, email and mobile phone number for E-Service, E-Mail and SMS, then our data structure takes this form (Figure 4).

![Figure 4. Conceptual data structure of the organizational index](image)

This way we can create n-way association index between everything in e.g. 109342 is on room 301.05 which is Software Development Office which one of employees is 109342, which in fact the above person. This way we will be able to query hierarchically higher and lower attributes of one concept with single search, in e.g. querying 070576668 will return only 109342, but querying 109342 will return also username = ve09342, email = evisar@gmail.com, phone = 070576668, room = 301.05, etc. This way we can collect as much relevant
information as we would like to. In figure (4), the phone model and operations system is in fact irrelevant which wouldn’t have been collected at all for this purpose, but in situations where secondary data gathering is important and can’t be performed automatically, we can allow our users to manually update some parts of the index, in the form of trees or mind mapping, or even create an information management job position which continuously updates data on the index by request of periodically.

For the purpose of this paper, and without getting out of the scope of this research, we will configure our system to only gather and return contact and membership information in e.g. 109342 he has the following contact information and is member in the following concepts (Figure 5).

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>109342</td>
<td>Visar Elmazi</td>
<td>109342</td>
</tr>
<tr>
<td>username</td>
<td>ve09342</td>
<td>109342</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:evisar@gmail.com">evisar@gmail.com</a></td>
<td>109342</td>
</tr>
<tr>
<td>phone</td>
<td>070-576-668</td>
<td>109342</td>
</tr>
<tr>
<td>room</td>
<td>301.05</td>
<td>109342</td>
</tr>
<tr>
<td>301.05</td>
<td>Software Development</td>
<td>301.05</td>
</tr>
<tr>
<td>employee</td>
<td>109342</td>
<td>301.05</td>
</tr>
<tr>
<td>class</td>
<td>CCS001/05</td>
<td>109342</td>
</tr>
<tr>
<td>301/05</td>
<td>Information Systems Lecture with Zamir Dika</td>
<td>CCS001/05</td>
</tr>
<tr>
<td>day</td>
<td>Monday</td>
<td>CCS001/05</td>
</tr>
<tr>
<td>time</td>
<td>12:30</td>
<td>CCS001/05</td>
</tr>
<tr>
<td>course</td>
<td>CCS001</td>
<td>CCS001/05</td>
</tr>
<tr>
<td>teacher</td>
<td>z.dika</td>
<td>CCS001/05</td>
</tr>
<tr>
<td>CCS001</td>
<td>Information Systems</td>
<td>z.dika</td>
</tr>
<tr>
<td>z.dika</td>
<td>Zamir Dika</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. Relational indexed data table

We predefine also a list of keys which should be considered as contact information in e.g. username, email and phone and we are ready to begin mapping. Considering the integration part, the data is going to be gathered from several databases, namely: contacts, people, students, courses, exams, buildings, projects, etc. With this configuration, if we would have performed an unconstrained query with the term 109342, we would have obtained any of the values contained in fields Key, Value or Reference, but if we ask only for contacts and specify what contacts in e.g. phone or email, than the list narrows only with that information. This way we can very fast and easily get the context of the target event before generating the multicast message.

3.3 Event queuing service
The event queuing service will register a list of permitted application or users allowed to add events and will collect those events in a queued fashion; that is, between the event collector and message delivery service we will configure a message queuing service which will process individual events and generate appropriate messages for each event. We have configured an XML web service application which will represent a façade to the event gathering and mapping service. The XML web service is cross platform and cross application compatible, so we guarantee that including the current systems and databases, any future system will be able to notify our service about events of any kind.

Application integration won’t be the biggest issue when compared to the event generation activity. Generating events for application that have already been developed, which for the best part of them we don’t have access to the source code or we can’t afford to modify, is the probably the biggest challenge in this effort, together with organizational mapping. In order to simplify this problem, we first try to separate the parts that are more easily solved, like list all the applications that run on relational databases systems that support update triggers and perform analysis of impact on implementing these triggers. List all applications that don’t support update triggers and explore ways of maintaining synchronization states of their primary keys in order to determine what records have been changed. This method may be somewhat slower than the first method, depending on the cycle of checkup a synchronization component will do, etc.

Generally, if we won’t be able to catch events when they happen, we will have to manually push events on the service through the XML web service by interfacing the web service to the human factor (Figure 6).

3.4. Event-organization data mapping
After catching and adding the event in the database and providing enough usable metadata, the mapping component is activated, which as a result returns the number of affected contacts, in e.g. usernames, emails or mobile phone numbers, by the given event. In order to provide convenient user interfacing and readability, some of the generated event message bodies can contain named fields that can be filled by the mapping component, at the time of actual message generation. Below we show a sample message that can include named fields such that it can easily map into a multicast...
message.

Dear {Forename} {Surname},

Professor {Exam.Teacher.Forename} {Exam.Teacher.Surname} has published grades for the final exam {Exam.Occurrence} – {Exam.Course.Name}, held on {Exam.Date}.

Your published exam grade is {Exam.Grade}.

Message sent on : {@Date}, {@Time}

Figure 7. Preparing user readable messages from generated events

In the example at Figure 7, we have added an event which notifies students that their grades have been published for a certain final exam. In order to generate a multicast message, we need first enough metadata to locate the actual exam and teacher in order to extract the list of affected students by that update. For this typical case, a possible metadata might be: exam.occurrence = CCS001/05 or simply exam (CCS001/05). Strictly said, this generates a query in the form of return all students who have participated in the exam CCS001/05, or in more technical terms: return the set of all username attribute values where siblings of type exam contain the value CCS001/05.

First, this only returns the nodes that comply to the requested query, but then the component does a second pass where it discovers all the required named filed values, used in the event message. This way we are able to intelligently discover all the relevant destination users to notify with a certain message and then modify the message to contain relevant personal or contextual data concerned with the destination user.

3.5 Message delivery service

After the mapping component have generated the list of recipients and customized the messages for each individual recipient, the list of all messages if generated and delivered to the message delivery service. The message delivery services than analyzes, upon predefined rules, what message delivery device is going to use to deliver the actual messages. Depending on the recipient, the service may for example send emails when reading list of emails, or send SMS messages when determining that the recipient address is a phone number or send E-Service notification when reading a valid SEEU username, etc. It’s out of the scope of this paper to discuss and analyze various delivery services and guaranteed message delivery. But nonetheless, we have chosen to use message queuing before delivering the message to the delivery devices, so that we can track all the outgoing messages, leaving our event notification system.

Sometimes, sending messages to devices or even sending e-mailing messages, may fail. Thus, we will first send notification to the E-Service and then proceed with device communication. This way we will guarantee that no matter what, at least the user will have an option to check notifications some later point in time.

3.6 Message delivery constraints

One very important aspect to note when delivering message to devices is the message length and encoding. Being aware that in our environment, the usage of Cyrillic encoding might be required, we will create interfacing systems which will require generating two message bodies, one that does not exceed the texting limit of 120 characters, 60 for Cyrillic encoding and the other without limits, for the purposes of E-Mail and E-Service. This way we will make sure, for the most part of delivered messages, that they are readable by the target devices.

4. Conclusions

A notification system which extracts the multicast semantics from organizational or operational data may shorten the information delay by many factors. If we consider the fact that our users frequently check E-Service and E-Mail for news and notifications, we may expect improvements in the actual quality and student satisfaction by several degrees at least to the first level of sigma from standard quality deviation, which means movement from the actual 34% to nearly 76%. If we also include the mobile phone interfacing (SMS) we can dare to conclude that we may expect 90% quality of service for notifications, which is near the second sigma level for quality and the overall student’s experience should be nearly three times better that the actual.

Future notification systems should include options for gathering organizational semantics, in order to increase the level of service and target users more intelligently rather than simply broadcasting notifications.

References


