

# IENS: An Intelligent Event Notifying System for Web Service Offered Information

Sirajum Munir  
University of Virginia  
Charlottesville, Virginia  
munir@cs.virginia.edu

Tanima Dey  
University of Virginia  
Charlottesville, Virginia  
td8h@virginia.edu

## ABSTRACT

*With the advancement of the Internet technologies, we get bombard every day with a lot of useful and necessary information generated through this enormous distributed system. So, it becomes very difficult to remain up to date with the latest necessary information. These information is available, but distributed in heterogeneous systems. Designing and implementing a system using interfaces of several heterogeneous systems is challenging. In the world of Internet these days, the interfaces to different web services and the associated protocols implemented are very diverse. That's why it is very difficult to pull some particular information out of a system utilizing these different interfaces to provide a uniform and single access point to different services offered by the Internet. At the same time, a general language for event description is hard, especially when the events are quite dissimilar in nature. In this work, we design the architecture of such a system and develop an intelligent event notifying prototype based on this. The system is able to use the interfaces for electronic mails (e-mails) and one social networking website, Twitter and pull relevant information as specified by the user. We describe the architecture in details in this paper along with a novel Event Description Language (EDL) that is able to characterize different features and characteristics of various events associated with different web services, e.g., an e-mail or message received from a particular sender, status update of a friend in a networking site, new update in a blog entry, updated weather report.*

## Keywords

Event Notification System, Web Service Information, Event Description Language

## 1. INTRODUCTION

Human mind remains pre-occupied with so many thoughts that the attention for some particular events or information may be missed out for such involvement. The events or

information can be very crucial for the person, but because of being under various circumstances and stressful life style, these can go without any attention. This might result in several losses, such as, financial, personal, or social. Also, the attention and characteristics of human mind has always been a challenging aspect in designing softwares.

In today's world, the Internet undoubtedly plays an important role which influences the lives of people at all ages irrespective of jobs, careers, positions in a company etc. All have something in their lives or works that are highly dependent on various Internet web services, e.g., electronic mail or e-mail, online banking, social networks, personal interests. People now-a-days read news and get weather reports from different websites rather than watching the television. Students and faculty members get important updates for their research from the vast collection of information provided in different websites and portals, such as, ACM, IEEE which maintain all the published papers in online databases. They also know about the research conducted by other people in other universities. Internet has improved the communication through instant messenger service, different social networking sites like Facebook, Twitter, different professional networking sites, such as, LinkedIn etc. Whenever anyone needs any information about anything, e.g., shopping, banking, research, studies and so on, s/he can simply query the desired keyword in any search engine which eventually generates a long list of search results from which the desired information can be retrieved easily. These search engines have become such an integral part of our lives that "google" has now become a buzzword. In a summary, Internet has made everyone's life easier from every aspect.

As a result of this technological blessing, people now-a-days manage most of their important works and daily activities through different services provided by the Internet. For example, exchanging emails for the purpose of business or communication, maintaining networks with other people using Facebook, LinkedIn etc., paying bills, maintaining bank accounts, doing research or even shopping. The Internet has become the new encyclopedia where every kind of information can be found. Coping up and maintaining coordination with the busy life, along with these different kinds of profiles/sources of information can be hard to track. For example, at a particular time, when a person is busy with work, might be interested only to receive an e-mail from a particular person or be notified the due date for paying bills. If s/he is too busy elsewhere, there should be some way to send him notification or alert that the expected e-mail has arrived or due date of paying bills is today.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WOODSTOCK '97 El Paso, Texas USA

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$10.00.

The Internet services are supported by a wide variety of servers. There are many variations in the core design and implementation of these servers even for the same service. These differ in communication protocol, implementation languages, system compatibility requirements etc. Again, when there are many types of events, a single specification describing all is challenging to find.

In this paper, we address the broad problem of putting these individual heterogeneous systems or servers under a uniform interface using a very general Event Description Language. The problem is crucial as everyday there are new websites providing information and different services that it hard for a person to keep track of the existing ones, as well as the new ones. There is always a need for a single point from which any kind of interesting information can be gained rather than maintaining different points to check. The uniform interface or system design incorporating such heterogeneous interfaces can be utilized by certain event notifying systems which provide the user a single interface to check for the desired web-events. In today's world, where everyone is so busy that even a single minute counts, such unifying system can save a lot of time and overall make people's lives much easier. Such notifying system can send alerts and messages to the users about deadlines, important e-mails, most recently updated information about anything the user might be interested into, through a single terminal so that even if the user is busy doing some other work will eventually get the alert and take appropriate action. These systems thus save the time of the users, help them not to miss a deadline/due date and make their lives less stressful and hassle-free. The solution can increase the promptness and efficiency of the people using the system. Such systems can also be used as a source of advertisement that can promote different products and help developing the economy of the country.

In this work, we provide a possible solution to this problem. There are existing interfaces to multiple web services to pull information from, such as, mail fetching interfaces from Google Mail (GMail) server, API for pulling information from Twitter etc. We propose and design an architecture for a system that utilizes these heterogeneous interfaces and develop an event notifying system based on this design. This intelligent event notifying system keeps track of every information provided by different web servers and is able to pull relevant information, as specified by a particular user of the system. The information might be related to any web services, e.g., an e-mail, blog entry, news, message, weather. The system in the broad sense is able to save event notification preference as specified by the users, search for the information in the event specification through the interface among various servers and at the end pulls the information and sends notification to that user. We also describe a general and novel Event Description Language (EDL) for such notification system that incorporates different characteristics and features of these web services.

The paper is organized as follows: Section 2 describes the related works, Section 3 describes the requirements of the possible solution to the problem described, Section 4 describes the proposed system architecture and the Event Description Language. In Section 5 and Section 6, we discuss the risks and evaluation of the system developed respectively. And in Section 7 we conclude.

## 2. RELATED WORK

There have been several works in event notifying systems of different types and approaches in both software industry and academia. In this section, we describe some of those works in small details.

In general, there are event notifying software implemented in several e-mail service providers and social networking site. For example, both Yahoo! and Google e-mail interfaces contain calendars and allow users to set events on any date so that they can be notified of the event on that day. There is also a Yahoo! Widget Netsuite Event Notifier which is an event reminder application built for Yahoo! Widget Engine and uses web services to obtain events from the NetSuite Calendar [13]. Similarly, Facebook has an application that reminds the users about their friends' birthdays and shows the list of friends and their corresponding birthdays in the home page. Both of the applications apply some event monitoring function that sends users the notifications.

Popular online shopping site Amazon maintains similar service for special accounts that sends electronic notification about order updates and order information to the users. It allows to enable third-party software applications to retrieve event information from users' account in a reliable and secured manner [2, 3]. There is another application called AlertSite that has a different purpose but uses similar approach. This application is mainly for ensuring that websites, web-services and applications for the signed up customers are always available and running at peak performance, otherwise the system sends alerts to them via cell phone or e-mail [1]. At [5], there is an online application that supports plug-ins for maintaining yearly, monthly, daily, even hourly event scheduling online and managing appointments.

There are several applications for notifying e-mail reception, e.g., MailBell[10], ePrompter[6], POP Peeper[12] for Windows, KBiff[8] for Linux, Mail Beacon[9] for Mac. All of these softwares basically incorporate and monitor multiple e-mail accounts of the users where each mail server supports different mail fetching protocols, e.g., POP3, IMAP. These applications provide a single interface to the users so that they don't have to check Inbox for each individual e-mail accounts. A similar website is mint.com[11] which provides the users a single interface to manage all bank and credit card accounts and does all the necessary book keepings. It helps the user to save time by not checking each and every online banking/credit card account. It takes care of all the credit card bills' due dates, balance information, maintains securely the credentials of each online account/profiles and also assists the users in budgeting. Our prototype does similar things, but rather than maintaining a single type of account, it incorporates, maintains and monitors various kinds of accounts, e.g., both e-mail and social networking site.

In [16], the authors extend the traditional three layers of web services into four layers and provides a framework called Web Service Notification Framework using software design pattern. This work described in the paper has been later extended and published as a book chapter [17]. In [14], the authors perform a comparative study between the webservice-eventing and webservice-notification for Grid Computing and describes their project WSMessenger [15] that supports both of these specifications and heterogeneous resources and provides a middle ground.

### 3. REQUIREMENT ANALYSIS

In this section we analyze the requirements of the Intelligent Event Notification System in a broad sense. We classify the requirements into functional and non-functional requirements.

The functional requirement is to allow the users to specify any possible events of interest, and to notify them in a specific amount of time in their preferred way. The event of interest can be an e-mail received from a special person, information/news updated/published in a particular website, or, events in social and professional networks through Facebook, Twitter, and LinkedIn etc. The user preferred notification type can be of various forms. It can be an e-mail, a message or phone call through wireless telephone service or a pop-up window in a personal computer.

As far as the non-functional requirements are concerned, the system has to be able to work in a heterogeneous environment, and adapt to the change in the environment. The security of the system should be maintained. The users must obtain proper authentication information during signing up before using the system. The usage of the system must be maintained by proper authorization. The privacy of the users should be supported so that no user can see the events set and authentication information used by the other users. All the information of individual users should be kept confidential. The system should be scalable so that it supports hundreds and thousands of users and maintain high availability as well. It has to be responsive and it has to offer its services in a cost-effective way.

Validating the requirements of software system is very crucial which states whether the right system is being built or not. We can validate the system by prototyping. We develop the system in Agile software development process in which we always have a working system in hand. We deploy the system after it is developed at the end of each iteration and get the feedbacks of the users after they have used the system. They can report us back what are the functionalities that have been missing from the system, what are the features that are incomplete, the easiness of the usage, their satisfaction about the system as a whole etc. Based on the users' feedback, we can modify the system in the next iteration to incorporate the changes necessary to build the right system.

There are two scopes of our project. In the long-term scope, we envision a system which is able to provide any kinds of information and event notification to the registered users as long as it satisfies the users' interests. We have to be careful enough not to send any unrelated information to the users. The long-term solution should be flexible and adaptable enough so that it can incorporate and support any change of interfaces of the services and web community. It should also be extensible so that new web services and functionalities can be added. It has to offer other non-functional requirements too.

In the short-term semester-long scope, we are considering the e-mail notification system in which the users are able to get notification through our system when a new e-mail arrives from a particular e-mail address. This e-mail address can be of that person's supervisor, advisor, family member, manager at the bank etc. In this version, we also incorporate the Text-to-Speech converter by which the system reads out the e-mails sent to the users. This requirement can be modified a little such that disabled (especially, blind) people

can stay in touch through e-mails or social networks. The notification that will be sent to the users will be in a form of pop-up window with set alarm, a message or call in the cell phone. We also pull information from one of the networking web services, Twitter, and send notification to the users who is following him/her, about any newly received messages or status update of a friend.

### 4. SYSTEM DESIGN

In this section, we discuss the challenges we face for designing the prototype of the intelligent event notifying system, our proposed design approach to overcome these problems and the Event Description Language.

#### 4.1 System Architecture

In this subsection, we describe our proposed architecture for an intelligent event notifying system. Note that based on our requirement analysis, our system must be able to work with heterogeneous systems, able to adopt to the changes in web services and interfaces, able to preserve privacy, have high availability, be scalable, minimize response time and offer services in a cost-effective way. By considering all of these requirements, we propose a *client-server architecture with implicit invocation*.

The process view of our proposed architecture is shown in Figure 1. A user uses his favorite web-browser or desktop application, we call it Client Application to log in to our system. At first, the user has to register and the registration information is stored in the Database Server (shown as DB). When a registered user logs in, the system uses Database Server to authenticate the user. Then the user specifies his events of interest and the way he wants to be notified by using Event Description Language (described in the next section), where the events are stored in the Database Server. After authentication, the Client Application gets connected to the Application Server and waits for notification of the specified events of interest.

The Mail Server is connected to different mail servers (like Gmail, Yahoo!) and periodically checks for mail related events of interest. When the Mail Server detects some events of interest to take place, it notifies the Application Server. Application Server interacts with Database Server and notifies the user based on user-specified notification approach, i.e., if the user wants to be notified through SMS, the Application Server sends an SMS, or if the user wants to get a pop-up message, the Application Server sends a message to the Client Application which shows the pop-up message to the user. The Web Server works in the same way as the Mail Server does, except that it checks for web related events of interest. Note that there may be multiple Web Servers needed, since the API offered by different web services are language and/ or platform specific, and to fetch information from those web services, we may need to develop more than one Web Server. Both the Mail Server and the Web Servers interact with the Database Server to recognize the events of interest. Note that Web Server and Mail Server use implicit invocation to notify events to Application Server and Application Server uses the same approach to notify the Client Application.

#### 4.2 Event Description Language (EDL)

The proposed Event Description Language allows a user to specify his events of interest. The events at which users

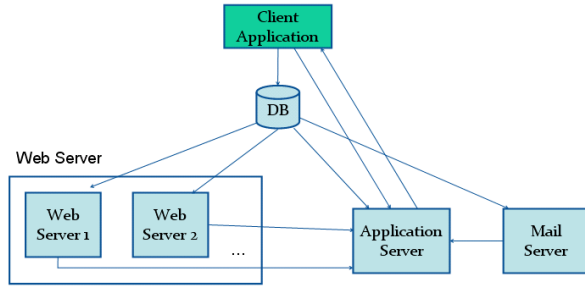


Figure 1: Process View of IENS-Architecture

might be interested in can vary a lot, and it is very difficult to come up with a language general enough that allows users to specify all possible events they may be interested in. The language specification is so challenging, it can itself be considered as a research problem. Our Event Description Language works by matching user-specified terms with the published text by web services which is almost similar to perform query in a search engine, like Google.

We define Event as a three tuple:

$$Event = \langle Category, Category-Spec-Info, Condition \rangle$$

*Category* can be either *Email*, if the user is interested in e-mail related events, or *Web* if the user is interested events published in some webpages:

$$Category = Email|Web$$

*Category-Spec-Info* is a 3-tuple which specifies the *Category* more specifically along with *UserID* and *Password*.

$$Category-Spec-Info = \langle Category-Name, userID, Password \rangle$$

For example, if a user is interested in email notification of his GMail account having user ID ‘munir’ and password ‘pass123’, the *Category-Spec-Info* will be  $\langle gmail, munir, pass123 \rangle$ . If a user is interested in web update notification of his Twitter account having user ID ‘tanima’ and password ‘pass456’, the *Category-Spec-Info* will be  $\langle twitter, tanima, pass456 \rangle$ . Note that this information is necessary to fetch information from web services.

*Condition* specifies the event of interest by using terms. A *Condition* is a sequence of ANDed 3-tuples.

$$Condition = \langle Type, Inclusiveness, Value \rangle AND \langle Type, Inclusiveness, Value \rangle AND \dots AND \langle Type, Inclusiveness, Value \rangle$$

For email related events of interest, the *Type* can be *Sender*, *Subject* or *Body*.

$$Type = Sender|Subject|Body$$

For web related events of interest, for example, a notification from Twitter, the *Type* can be *Message* or *Status*.

$$Type = Message|Status$$

*Inclusiveness* specifies whether the *value* should be or should not be within the published information. So, it becomes

$$Inclusiveness = Has|Does-Not-Have$$

*Value* specifies one or more words that will or will not be within the published information to consider it as an event of interest. It can be thought of a keyword used to search for in some text.

For example, if a user wants to get notified when he receives any email from his supervisor having email address ‘supervisor@gmail.com’ about ‘project’, the *Condition* may be  $\langle Sender, Has, supervisor@gmail.com \rangle AND \langle Body, Has, project \rangle$ . The user may want to get notified for any email related to Christmas, for example, emails having the word ‘Christmas’ in the subject or in the body. Since Event Description Language does not support ‘OR’ in specifying *Condition*, he has to specify two different *Events* by specifying two *Conditions*  $\langle Subject, Has, Christmas \rangle$  and  $\langle Body, Has, Christmas \rangle$ , one for each event. Also, if a Twitter-user wants to get notified when any one is saying anything about ‘Alice’ in their status, he may specify the *Condition* as  $\langle Status, Has, Alice \rangle$ . If he wants to get notified for any private message in his Twitter account about ‘travel’, he may specify the *Condition* as  $\langle Message, Has, travel \rangle$ . Note that the type of an event, whether the it is a web-related event or a email-related-event and the domain where it is expected to take place like GMail, Twitter etc. can be determined by *Category-Spec-Info*.

It is obvious that we cannot specify all possible events by using this Event Description Language. But it can serve as a baseline solution and its functionalities can be further improved by adding more options in the *Type* field.

## 5. RISK ANALYSIS

In this section, we identify a number of risks that may jeopardize the successful implementation of the project. We also specify the way we are addressing these risks. The major risks are described as follows:

1. **Accessing Web Servers:** Different web servers have different language-based-APIs. So, we may have to develop multiple servers to get information from different web servers. The web servers may release new version of their APIs and may not support the previous versions. In that case, we have to update the implementation by using the new version of the APIs.

To address this risk, we tried to connect to Facebook server to fetch information. But the problem is, there are lots of conflicting statements about the feasibility of creating a single desktop application that will directly interact with the Facebook server [7]. The main complaint is, a user has to go to a web browser to do the login, then come back to the application to interact with the Facebook server for security purpose. It can

be implemented in C#[7] without this problem, but it is not clear whether other language-based-APIs actually offer this opportunity. An alternate approach may be using callback URL with some web server. At the time of creating the Facebook application, you have to enter a callback URL, which is the address where your application lives on your server, or the server where the application is being hosted. We can build a web application running on a web server that will directly communicate with the Facebook server and will work as a client to the Application Server (specified in section 4.1) and notify the Application Server when an event of interest takes place, which is possible, but more time-consuming. Also, there is a lot of complaints [4] and bug reports about the available Facebook APIs. So, we decided to work with Twitter instead of Facebook for this short time project and used Java Twitter API to connect to Twitter server. We have successfully fetched the direct messages and status updates of the followers, followings and friends of a Twitter-user. Note that the implementation has to be very efficient because Twitter API performs account and IP based rate-limiting and allows a client to make at most 150 requests per hour. Thus our prototype implementation shows that we can take a similar approach to interact with other web servers.

2. **Accessing Mail Servers:** Accessing mail server requires mail server specific information. For example, to access GMail account using POP3 protocol, the system has to know that the host name to be considered is pop.gmail.com where as if the user prefers to access his account through IMAP protocol, then the host name will be imap.gmail.com. Also, GMail has a settings page that control which messages are available to the user through which protocol. If the settings are not properly set, the system will not be able to fetch mail even if it knows the mail server specific information. The major risk is, the user may have mail account in an arbitrary mail server, and he may not know these mail server specific information.

To address this risk, we are limiting our service to some specific mail users. For example, for the initial phase, we are only supporting to the Gmail users for their mail related notification. We can extend our services based on the requests from the users. Alternately, we can allow the users to enter the mail server specific information. This may be useful for some advanced users.

3. **Specifying Events of Interest:** Specifying events of interest is very a challenging task since users may be interested in any possible event and some events may be even difficult to express. This is also a very crucial part of the system. Because, if the users are not able to specify their events of interests, they may not be interested to use the system any more.

To address this risk, we designed an Event Description Language (EDL) (specified in section 4.2) that allows a user to specify events of interest customized for available domains. For example, if the user is interested in email related notification, the EDL allows to define events of interest based on the sender's email address,

or specific words in the subject, or in the body of the email. Similarly, EDL allows a Twitter-user to define events of interest based on words in his friends' statuses or in the direct messages sent to him. Currently our EDL allows limited type of events to be specified, but it can be extended by adding more types in the condition types.

4. **Deadline and Other Issues:** We have a very close deadline (around 8 weeks) to complete the project, at least a major portion of it to demonstrate the feasibility of the whole project. We are only a two person team. At the beginning, we did not have any idea which platform we should use, which APIs are available etc. We had to try different approaches to address the major risks. For example, to demonstrate the feasibility of interacting with a popular web server we selected Facebook at the beginning, but we had to move to Twitter because it would take longer time to demonstrate the feasibility using Facebook.

## 6. EVALUATION

In this section we evaluate our approach to check if it addresses the risks adequately and satisfies our project goals. Let's revisit what we are claiming to offer. Our system has to be able to work in a heterogeneous environment, adapt to change in web services, preserve privacy, have high availability, be scalable, minimize response time and be able to offer services in a cost-effective way.

The evaluation is based on implementation experience and example. We have designed the system architecture in a way that it will be able to work in a heterogeneous environment. Because, the Web Servers can be developed in different languages and/ or platforms, and the Mail Server, Application Server and Client Application are separated and communicate each other through socket connection that makes it easier to develop and work in a heterogeneous environment. The adaptation to change in the web services can be overcome by designing classes using Adapter pattern. So, if the application needs to be updated by using a new version of API, we don't need to change the whole application. Instead, we will only replace some driver class for that application domain, and the rest of the code will remain the same. The system will be able to preserve privacy by using standard cryptographic techniques. To offer high availability we need to add redundant servers and distribute load among the servers. The system is scalable since we are using implicit invocation in this client server architecture. So, the Application Client or Application Server does not need to periodically perform query to the Mail Server or Web Servers for any latest events of interest. To minimize response time, we need to deploy more number of servers and distribute load among them. We can design the system to gain some profit too. For example, we can offer different types of services with different charges. If a user has to be notified in a shorter amount of time, he will be charged more during the sign up phase. Designing the system in this way will allow the system to gain profit as well as ensure its sustenance. So, by developing a prototype of the system and by demonstrating these techniques that people have been using for years, like design patterns for adaptation, cryptography for preserving privacy, redundancy and load balancing for availability and scalability, different services for profitable

business model, it can be stated confidently that it is very possible to develop an intelligent event notifying system and achieve all of these goals successfully.

## 7. CONCLUSION

In this work, we propose, design, and validate an Intelligent Event Notifying System that allows users to specify their events of interest and notifies them in their preferred way. We strongly believe that this service will significantly improve people's activity and save a lot of precious time.

## 8. ACKNOWLEDGEMENTS

We would like to express our gratitude towards our course instructor Dr. Kevin Sullivan for his valuable feedback and advices. We would also like to thank Kirti Chawla for providing us the library for sending SMS which is used in our prototype event notifier.

## 9. REFERENCES

- [1] Alerts site. <http://www.alertsite.com>.
- [2] Amazon.com: help on managing orders. [http://www.amazon.com/gp/help/customer/display.html/ref=hp\\_bc\\_nav?ie=UTF8&nodeId=13324351](http://www.amazon.com/gp/help/customer/display.html/ref=hp_bc_nav?ie=UTF8&nodeId=13324351).
- [3] Amazon.com: help on managing orders. <http://www.amazon.com/gp/help/customer/display.html?ie=UTF8&nodeId=200122950>.
- [4] Bugs at facebook java api. <http://t-machine.org/index.php/2007/09/21/ten-tips-and-tricks-for-writing-facebook-apps/>.
- [5] e-information organizer. [http://einformationorganizer.com/?a=BIMS\\_Organizer\\_Schedule\\_Manager](http://einformationorganizer.com/?a=BIMS_Organizer_Schedule_Manager).
- [6] eprompter: Free email notification software. <http://www.eprompter.com/index.htm>.
- [7] Facebook log-in issue. <http://aavatar.digital-radiation.com/blog/?p=505>.
- [8] Kbiff - kde mail notification utility. <http://www.kbiff.com/>.
- [9] Mail beacon. <http://brainaware.com/mb>.
- [10] Mailbell: Email notifier for windows. <http://www.emtec.com/mailbell/>.
- [11] mint.com. <http://www.mint.com/about/>.
- [12] Pop peeper: Email notification. <http://www.poppeeper.com/>.
- [13] Yahoo! widget netsuite event notifier. <http://suitesource.netsuite.com/s.nl/it.A/id.58/.f>.
- [14] Y. Huang and D. Gannon. A comparative study of web services-based event notification specifications. In *ICPPW '06: Proceedings of the 2006 International Conference Workshops on Parallel Processing*, pages 7–14, Washington, DC, USA, 2006. IEEE Computer Society.
- [15] Y. Huang, A. Slominski, C. Herath, and D. Gannon. Ws-messenger: A web services-based messaging system for service-oriented grid computing. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pages 166–173, Washington, DC, USA, 2006. IEEE Computer Society.
- [16] B. Kalali, P. Alencar, and D. Cowan. Wsnf: Designing a web service notification framework for web services. In *International Workshop Web Services Research, Standardization, and Deployment*, pages 166–171, 2002.
- [17] B. Kalali, P. Alencar, and D. Cowan. *NSPF: Designing a Notification Service Provider Framework for Web Services*, volume 2593/2009. SpringerLink, 2003.