ABSTRACT

JAIN, NEHA. A Web Services Based System for Parsing and Managing Calls for Papers. (Under the direction of Dr. Peter R Wurman).

Academic and industrial conference organizers invite members of academia and industry over e-mails containing Calls for Papers(CFPs). These invites are manually typed e-mails and do not follow a standard template. Currently, there does not exist any simple way of systematically condensing the conference data available from these e-mails. Hence, organizing all the CFPs and extracting each conference information from the e-mails involves manual parsing for key data fields such as event URL, submission deadlines, updates and so on. Also, the conference organizers have to maintain conference web pages and e-mail address lists to send the calls for papers, updates and reminders.

To simplify this tedious manual task, a system with automated data orchestration has been proposed. The system comprises of two key components. The first component, CeParse, is a parser with a pattern matching component that accesses CFP e-mails on the mail server, parses them and extracts the CFP data. This data is then forwarded to the second component,CReWS. CReWS is a Web service oriented central repository that accepts and displays the CFP data. This integrated system thereby eliminates the repeated manual overhead of parsing and updating the conference CFP data. Data from the repository is available for public viewing via a graphical interface with capabilities of sorting, searching and providing RSS feeds. This implementation offers a loosely coupled service oriented architecture which can serve to the advantage of operating with other software systems.

A Web Services Based System for Parsing and Managing Calls for Papers

by

Neha Jain

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh

2006

Approved By:

Dr. James Lester

Dr. Munindar P. Singh

Dr. Peter R. Wurman
Chair of Advisory Committee

a step closer to adding all colors to the rainbow of aspirations of my mother

a step closer to realizing my own dreams

Biography

Neha was born in India, in a quiet city named Ranikhet - The Queen of Fields. She spent her childhood with her family and then moved to a boarding school in Delhi, India, when she was 12. She completed her undergraduate degree of Bachelor of Information Technology from Delhi University, Delhi. Her years away from home have taught her a lot. Self reliance and confidence top the list. She continues to fuel it by challenging herself with as many things as she can. A Master of Science in Computer Science is the beginning. Dancing and driving are her passions. Music is something that she cannot do without.

Acknowledgements

I thank Dr. Wurman for his guidance throughout this work. He was on a sabbatical while I worked on my thesis, but he still found the time to help me and give me his invaluable advice. The unshakable confidence of my family when I needed it most gave me endurance. Fierce support of my friends was relentless. You helped me accomplish this. Thank you.

Contents

List of Figures

List of Algorithms

Listings

List of Tables

Chapter 1

Introduction

1.1 Background

The number of conferences, workshops, journals and other avenues that invite papers for publication is increasing over time. Researchers from all over the world, including academia and industry, are invited to submit their work for review for publication and presentation in such events. The invitations, or calls for papers (CFPs), are generally sent out to the invitees over e-mails that contain all the details of the upcoming events. The details include the event name, the submission and notification deadlines, event dates, venue, event URL and so on. The CFP notifications are often posted on web-sites maintained by the conference hosts. The researchers look for the invitations of interest, keep track of the required deadlines and submit their work for review to the conferences accordingly. The work is then reviewed by the hosts and they notify the researchers of the acceptance results. The invitees remember the dates of the conference that they have to finally attend.

The hosts of the events locate and keep track of all the addresses of the people and the mailing lists that have to be invited. They send out e-mails with reminders or schedule updates. They also maintain the web-sites where they host the conference information and invitations. On the other hand, researchers maintain their calendars with submission deadlines or the dates of the events of interest. They regularly check event home pages and monitor their e-mails for updates. This complete scenario is illustrated in Figure 1.1.

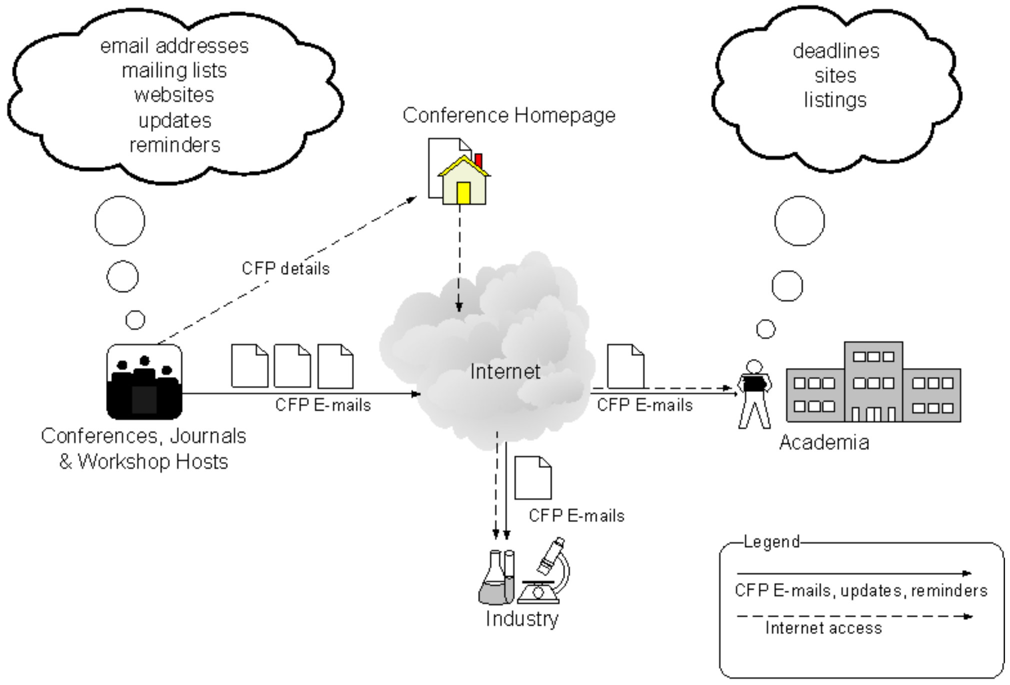


Figure 1.1: Background

1.2 Motivation

In the scenario described above, researchers find it difficult to systematically track CFP details conveyed in the e-mails. They can miss the opportunity to submit papers because of several reasons, including work pressure due to deadlines that are in close proximity, lack of proper documentation of CFP deadlines, absence of reminders for conference dates. Also, some e-mails are sent in formats that render them useless for the receiving e-mail clients. Further, some e-mails have attachments which might be ignored. As researchers receive several CFPs spanning varied research areas, there are chances of relevant CFPs getting overlooked along with the irrelevant ones. Also, it is a very time consuming task to read the entire mail only to find out that it conveys an update. To add to the overhead, the updates might be buried deep down in the body of the e-mail. Along with sending the updates, the event hosts not only have to update their address lists but they also have to monitor bounced e-mail invitations. Further, the invitations that are sent only via e-mail, are not available for the worldwide academic community as they are restricted to the people who are on the mailing lists of the hosts.

In such a situation, a central condensed listing of upcoming conferences would be a convenient tool for researchers. They will be able to access all the details and updates of CFPs at a single location. A number of such listings are maintained. One such example is a list of e-commerce related CFPs maintained by Wurman at the SIGecom web-site[1]. He receives CFPs from several sources and he short-lists the relevant ones. Then, he manually parses the short-listed invites for event details and posts them on the web-site. However, the combination of a hectic schedule and the tedious process involved in manually parsing the CFP data does not let Wurman update the CFP listing any longer.

This situation clearly demonstrates the need of a system that can automatically maintain an updated list of CFPs for upcoming conferences, workshops and journals along with an indication of the area of publication. Another prime requirement would be prompt regularity in updating the listing. Also, the event organizers should be able to add CFPs to this listing easily to have a far reaching impact of their invitations, updates and reminders.

This is the motivation to formulate a central online repository of CFPs which facilitates easy and continuous access, updating, and entry of CFP e-mails from all over the world. Since different researchers have different areas of interest, using a single view of the repository for all the CFPs is not the best solution. As in that case, the researchers would have to again look through all the CFPs to find the invitations of their interest. Hence, we need a solution wherein researchers can filter relevant CFPs for their own use or to publish on their personal web pages. They should also have an option of receiving updates through mechanisms like Really Simple Syndication (RSS) feeds, etc. We designed such a system in our work and present the features and implementation details here.

1.3 Contributions

Based on the problem and the implemented system, we make the following contributions through this work:

1. *C*FP *Re*pository *W*eb *S*ervice (*CReWS*), a loosely coupled Web services oriented system that accepts CFP data (such as event name, deadlines, URL and so on) via exposed Web services. It provides the following functionality:
    - Exposes Web services to accept *correct* CFP data and *questionable* CFP data and stores them in the *C*orrect *R*epository and *D*oubtful *R*epository respectively.
    - Exposes Web services to read data from both these repositories.
    - Detects updated or duplicated CFPs.
2. *CeParse*, a *C*FP *E*mail *Parse*r, implemented in Python that offers the following functionality:

- Parses the received e-mails for CFP data.
- Identifies CFPs into three categories based on extracted CFP data. They are: correctly parsed e-mails, questionablely parsed e-mails and non-parsable e-mails.
- Invokes the appropriate Web services exposed by CReWS to post the extracted CFP data.

3. Further, a Graphical Interface (GI) comprising of two components is described. The first component is the Graphical Administrative Interface (GAI) that can be used by the system administrators to edit the questionable data in the questionable repository. The second component is the Graphical User Interface (GUI) that can be used by hosts and to enter CFP data to the correct repository. Both these components also offer a complete listing of the CFPs in the correct repository.

1.4 Thesis Organization

In Chapter 2, related work for maintaining a central CFP repository and parsing of e-mails is reviewed. The contributions of this work are also mentioned. Chapter 3 illustrates the architecture of the proposed system. Chapter 4 explains the working of CReWS and the graphical user and administrative interfaces. Chapter 5 provides the overview and explains details of CeParse, its different entities and components. The system is evaluated in Chapter 6, followed by the discussion of future work in Chapter 7. Lastly appendix A lists the guiding patterns for CeParse and Appendix B lists the schema of the database that stores the CFP data at CReWS. Appendix C gives an overview of Web services technologies.

Chapter 2

Related Work

Event organizers use a variety of ways to invite academic and industry researchers to the conferences, workshops and journals. We look at the effectiveness of existing methods to detect areas of possible enhancements.

2.1 Current Solutions

We surveyed the existing systems and methods used by event hosts to send their CFPs to researchers. We classify the methods in three broad categories. The first includes the listings of CFPs maintained on various web-sites of the events themselves. The second method is to send CFP e-mails to people or to mailing lists. These recipients can include editors who manually parse information from these e-mails and either display it on their web-sites or further send it to mailing lists. The third method is to submit the details of the upcoming events to web-sites that explicitly gather the CFP information of several events and publish it for public viewing. Figure 2.1 graphically displays these methods. We examine the features and limitations of these methods along with their examples.

2.1.1 CFPs posted on an event web-site

Most of the organizers of workshops, conferences and journals maintain web-sites where they list the upcoming event details, updates, etc. These web-sites have to be regularly updated by the organizers. Researchers have to regularly monitor these web-sites for any updates or changes. Clearly, it creates overhead that could be avoided if there was a mechanism to inform the researchers

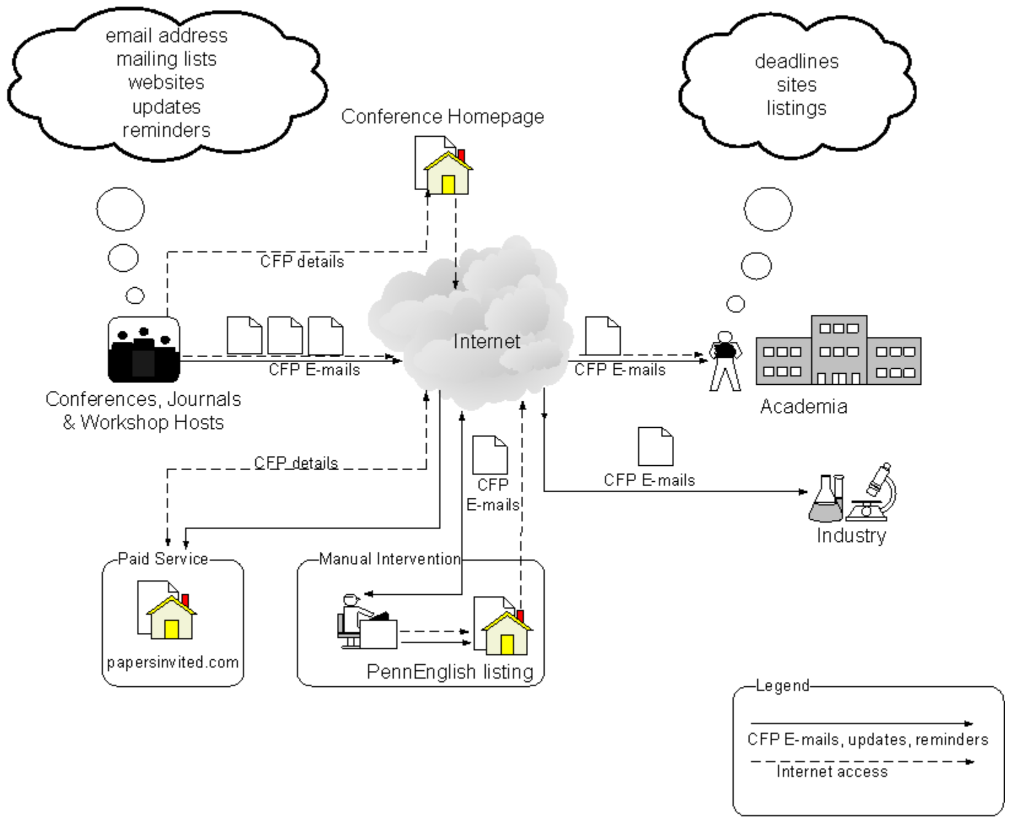


Figure 2.1: Current Scenario

of the updates.

2.1.2 CFPs e-mailed to researchers

Organizers of the events send CFP e-mails to researchers or mailing lists. The recipients of these e-mails can be researchers who monitor these e-mails for themselves. As mentioned before, these recipients can also include editors, who extract the CFP details from relevant e-mails. These editors generally use two methods to further promote the CFPs. We look at these methods along with examples to understand these types of announcements.

1. Manually parse CFP data and post on web-sites.

   Editors can manually parse the CFP data from the e-mails and post it on web-sites. One such example that we mentioned in Chapter 1, is that of the web page on the SIGecom web-site maintained by Wurman.

2. Manually verify the relevance of a CFP e-mail before forwarding it.

   Editors can manually short list relevant CFPs of interest from all the received e-mails. They can then forward these short-listed CFPs to mailing lists or post them on a web-site. One such example is PennEnglish[2], developed in 1995 by the English Department at the University of Pennsylvania for English and American Literature and Culture CFPs. The system is a combination of an electronic mailing listing and a web-site that lists the CFPs. They list the e-mail address and the format for sending the CFPs on the web-site itself. Anyone can post CFPs to the list using the listed e-mail address. However, the CFP e-mails or updates are manually approved by an editor and are then sent to the mailing list. Manual intervention involves the tedious task of the editor going through each CFP to check its relevance. It can introduce delays in postings as well. The people on the mailing list receive all the CFPs for English literature as they can not filter the posts to specify their field of interest. The approved CFP e-mail is also posted on an archive displayed on the web-site. The archives can be sorted by CFP topics. The archives are based on hypermail[3], which generates listings from the e-mail body.

2.1.3 CFP data posted on external web-sites

Event organizers can enter the CFP data on web-sites that obtain data from several event hosts and list them together for public viewing. One such example is the PapersInvited site[4]. It is a membership based web-site powered by CSA[5], a world wide information company. It exposes a graphical user interfaces where organizers can enter CFP data. This data is then displayed on the web-site along with the data of all other CFPs. Members can personalize their accounts and search through an exhaustive listing of CFPs across several domains. This web-site also lists an e-mail address where CFPs can be sent so that they can be listed on the site.

However, the problem of researchers receiving several CFPs that they have to parse through and filter for their own consumption still remains. They have to check the conference home pages or subscribe to a service like papersinvited[4]. PennEnglish[2] lists only the CFPs for English literature. Hence, we propose a listing of CFPs which has access to all CFPs received through e-mails. It can list the CFPs in a searchable and sortable format. This central listing can be customized as per user needs and can short list CFPs as per user interests.

2.2 Proposed Solution

As we illustrate in Figure 2.2, a promising solution would have the ability to effectively parse each human written CFP for event data on its own. It would store the data in a data repository that is accessible by all via a graphical *central listing.* The data repository would also offer easy methods to input CFP data, such as Web services or a simple Graphical Interface. It would accept edits to the data. The graphical listing would offer the capability to sort the displayed list on each field of the data and would be customizable to suit user needs.

Also, the researchers should be able to receive updates from this repository. So in effect, the organizers will just have to worry about sending e-mails to one e-mail address (that the parser accesses) or to send the data and updates to one central repository. The researchers will have to just track one central web-site and can concentrate on their work.

So, from this discussion, we identify three prime features for our automated solution. They are listed here:

- Universally accessible data repository.
- Extraction of CFP data from e-mails.
- Graphical listing of relevant CFP data.

These requirements encourage us to develop the automated solution based on an architecture that we explain in the next chapter.

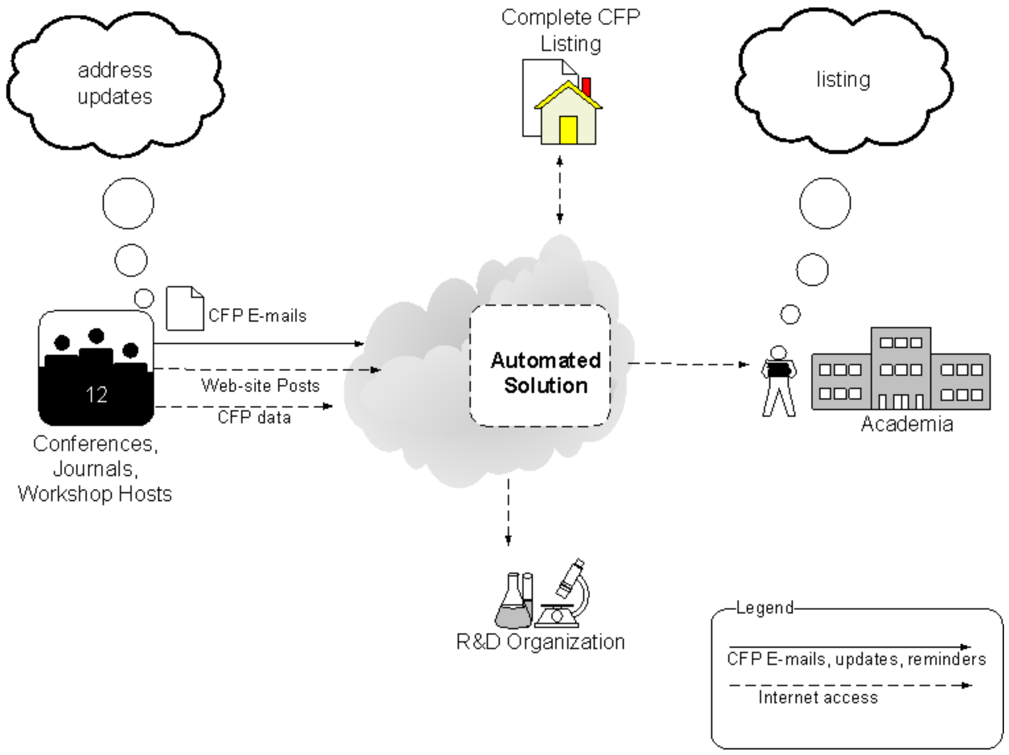


Figure 2.2: Proposed Solution

Chapter 3

System Architecture

We introduce the architecture of the new automated solution in this chapter. We also describe how each component in the architecture interacts with the other components to achieve the desired functionality of the system as a whole. The underlying technology used for developing each component is also explained.

The system consists of three main components: CReWS, CeParse and the GI, as illustrated in Figure 3.1. All the three components can be used to input CFP data. Depending on the method used, data from all the three components is stored in either the *correct repository* or the *questionable repository*, both of which reside with CReWS. The graphical interface then lists the CFP data available in the correct repository via the GUI (Graphical User Interface) on the central listing. The GI also provides the option of editing the CFP questionable data in the questionable repository via the GAI (Graphical Administrative Interface). We discuss these components now.

3.1 CReWS

CReWS is a Web service oriented global interface which provides the functionality of reading and writing CFP data from the data repositories hosted by it.

It provides read and write access to the CFP data repositories in two ways:

1. JSP/Servlet interface that makes local calls to java implementation. Currently, in our implementation, the Graphical Interface (as explained in the next section) accesses the content of the data repositories using this method.
2. By using exposed Web services that can be invoked by clients.

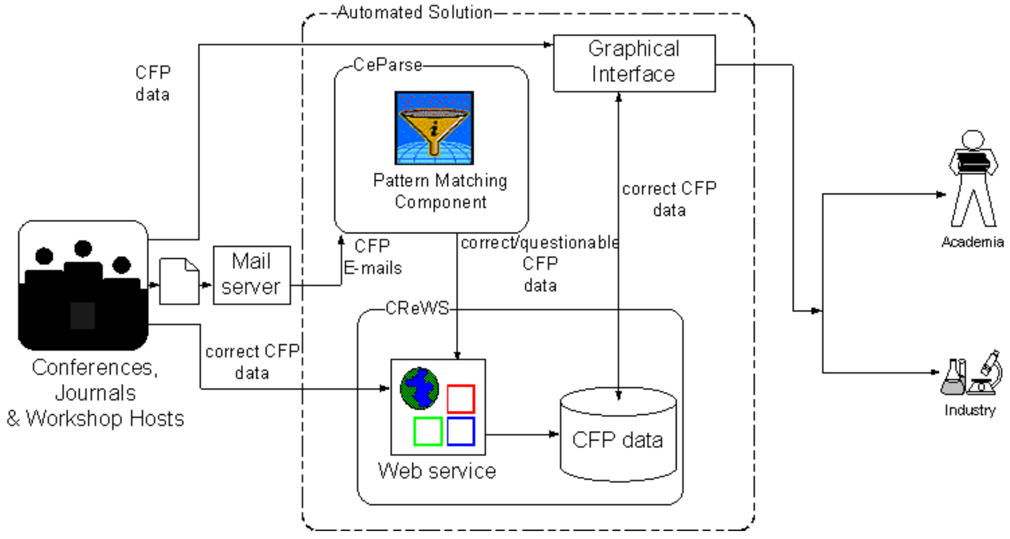


Figure 3.1: System Architecture

The services that are exposed by CReWS are:

- Read/Write correct CFP data to the correct repository.
- Read/Write questionable CFP data to the questionable repository.

Users wishing to send data to the correct repository or questionable repository can invoke these web services by accessing the exposed WSDL. They can develop their own customized client side code. A prime example of a such a client utilizing these services is CeParse, as explained later in this chapter. Another example of such clients are web browsers, that can invoke these Web services via JSPs, servlets etc.

3.1.1 Why Web services for CReWS?

We first look at the requirements that CReWS has to fulfill to achieve the desired functionality.

- Universally accessible global repository.

  It should serve as a global repository accessible to any academician in the world. He should not have the need to implement his own repository or maintain of his own. This provides reusability.

- Inter-operable implementation.

Anybody should be able to develop a web client or a parser client which can send the data to the repository in a pre-defined format. The client can be a .NET, C, C++, Java, Python, JSP/Servlet implementation and it should still be able to access the central repositories. This provides inter-operability.

These requirements are a big motivation to develop CReWS as a Web service as we see in the following sections.

Overview of Web services

Primarily, Web services combine the power of two widespread technologies, namely XML and HTTP, wherein one defines the data description and the other, transport. They provide loosely coupled functionality and can get isolated applications to share information, which is what we are looking for. The W3C's Web Services Architecture Working Group has jointly come to agreement on the following working definition of a Web service:

"A Web service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols"[6].

Their prime features are:

- Web services are modular applications that can be described and invoked over the internet. They are
  - Standards based
  - Inter-operable
  - Selfcontained
- Web services can be new applications or just wrappers around existing legacy systems to make them internet enabled.
- Web Services can rely on other Web services to achieve the goals and participate in business processes.
- Web services are called using SOAP and XML based requests.
  - Allows connecting heterogenous systems
  - Web services provide loose coupling implying compatibility with other software systems.

In our system, it is the SOAPy[7] toolkit that helps CeParse to invoke CReWS to send the CFP data to the data repositories. We further leave our implementation open to working with other software systems. It could be another parser like CeParse or any other software system that organizes and formulates conference details. Anybody can develop their own client-side code and interact with the repository at CReWS without any changes to CReWS.

A more detailed explanation of Web services is added in Appendix C.

3.2 CeParse

CeParse is a pattern matching component that accesses a pre-specified mailbox on an IMAP server. It accesses e-mails in the mailbox, parses them on the basis of certain patterns to extract CFP data, and forwards the data to the repositories on CReWS. It is guided by two sets of patterns, the *guiding patterns* and *classification patterns.*

Guiding Patterns The guiding patterns define the lists of words that the parser searches for in e-mails when looking for CFP data fields. For e.g., while looking for the field of dates in CFP e-mails, the parser searches for the date patterns listed in Appendix A. We describe these patterns in detail in Chapter 5.

Classification Patterns The classification patterns define the rules that the parser follows to classify an e-mail as a correctly parsed e-mail or a questionably parsed e-mail, based on the data extracted from the e-mail. For example, a null value of the event name in the extracted data, categorizes the e-mail as a questionable e-mail. We describe these patterns in detail in Chapter 5.

Helped by the guiding patterns, CeParse extracts data fields from all the e-mails on the mailbox. Based on the extracted data and the classification patterns, it classifies the e-mails in three categories. The three categories are:

1. Correctly classifiable CFPs

   The extracted data from these e-mails is the CFP correct data.

2. Questionable CFPs

   The extracted data from these e-mails is the questionable data.

3. Unclassifiable CFPs

   No data is extracted from these e-mails successfully.

Once this classification is complete, CeParse sends the data for the first two categories to CReWS. The correct data is sent to the correct repository while the questionable data is sent to the questionable repository by invoking appropriate Web services. These are exposed by CReWS (as mentioned earlier in this chapter) to accept CFP data. Hence, CeParse is actually a way to obtain CFP data from the CFP e-mails.

CeParse has been implemented in Python[8],[24],[22] and also uses the SOAPy toolkit to interact with CReWS Web services. The motivation behind using these technologies is described in the next section.

3.2.1 Why Python for CeParse?

Python is an interpreted language that has strong list and string manipulation features that are useful for the tasks performed by CeParse. Python works well with *regular expressions*[9],[21], that are used by CeParse to parse the e-mails written by humans. Python is easy to learn and the code is easy to read for anyone wishing to tweak the parser. It works well with Web services as well as mail servers without involving much extra effort.

The SOAPy toolkit provides a SOAP/XML schema library for Python. In our case, given the Web Service Definition Language (WSDL)[10] exposed by CReWS (explained in detail in Chapter 4), SOAPy APIs can invoke the Web services exposed by CReWS and hence the functionality of CReWS is available to Python quite transparently.

However, Python is not the fastest language in terms of execution speed. Because we receive about 6−7 e-mails per day and we want higher accuracy in terms of obtaining the right CFP data, run-time efficiency is relatively low on the priority list. These e-mails will be parsed automatically and would not require manual intervention. So a slow execution time is acceptable.

3.3 Graphical Interface(GI)

The Graphical Interface is a browser interface that is visible to all the hosts to enter CFP-data and for all to access the data in the databases. It is comprised of two parts:

1. Graphical User Interface(GUI)

2. Graphical Administrative Interface(GAI)

They are further explained below.

3.3.1 Graphical User Interface(GUI)

A browser interface through which the conference hosts can enter or read the CFP data. Functionalities provided by the GUI are listed here:

- Serves as an interface to enter CFP data into the correct repository.
- Serves as an interface to display CFP data from the correct repository.
  - Offers short listing of the displayed CFPs on the basis of keywords, for example: agents, Web services and so on.
  - Offers a sorting capability for the CFP listing on any data field. For example, submission deadline.
- Provides the functionality of subscribing to RSS feeds.

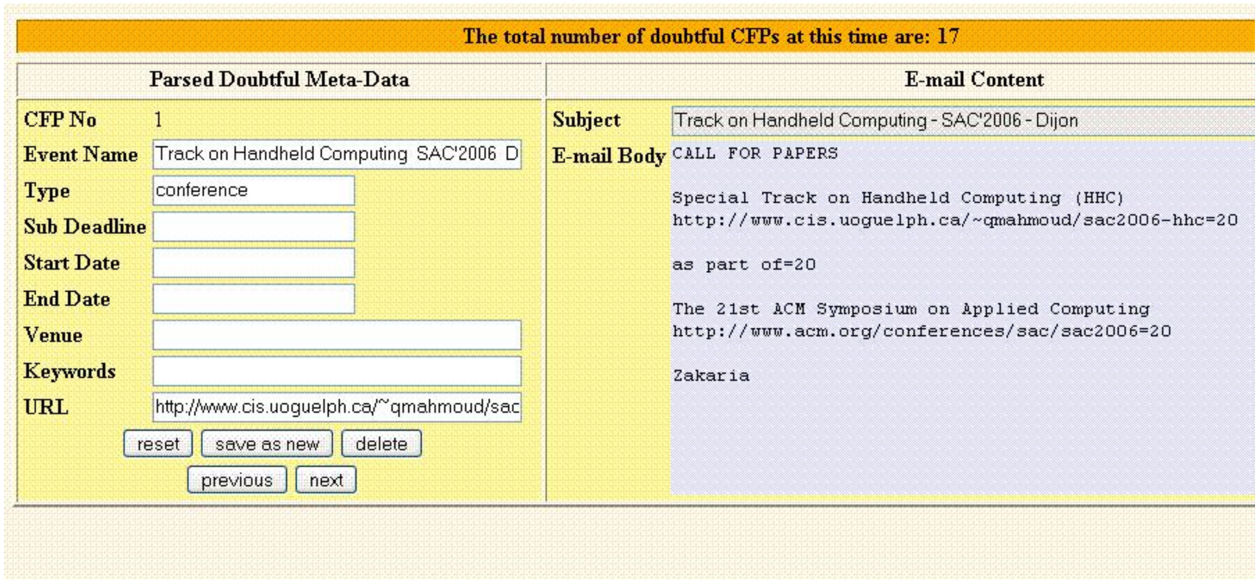
3.3.2 Graphical Administrative Interface(GAI)

The administrators of the system can access the CFP data in the questionable repository, along with the CFP data in the correct data repository with this interface.

The functionality provided by the GAI is:

- It lists the CFP data in the correct repository.
- It allows the administrator to compare the questionable data in the questionable repository with the correct data of any similar CFP in the correct repository.
  - If a similar CFP exists, it lists the similar correct CFP data along with editable questionable data of the questionable CFP. The administrator can review both the CFPs and decide if the similar CFPs are just duplicates or different CFPs. If they are duplicates, he can choose to discard the questionable CFP or if they are not duplicates, he can edit the questionable data and add it to the correct data repository.
  - If a similar CFP does not exist, it lists the questionable data in an editable format along with the body of the questionable CFP. The administrator can make changes to the questionable data and add it to the correct data repository or choose to discard it.
- It sends e-mail reminders to the administrator with a number of pending questionable CFPs.

Figure 3.2 shows a snapshot of the GAI. It is an instance where a questionable CFP has been displayed along with the actual e-mail body, so that the administrator can edit the CFP data and move it to the correct repository. The e-mail is a questionable case as the submission deadline and the start date were not parsed.


The total number of doubtful CFPs at this time are: 17

Parsed Doubtful Meta-Data	
CFP No	1
Event Name	Track on Handheld Computing SAC'2006 D
Type	conference
Sub Deadline	
Start Date	
End Date	
Venue	
Keywords	
URL	http://www.cis.uoguelph.ca/~qmahmoud/sac

E-mail Content

Subject: Track on Handheld Computing - SAC'2006 - Dijon

E-mail Body:

```
CALL FOR PAPERS

Special Track on Handheld Computing (HHC)
http://www.cis.uoguelph.ca/~qmahmoud/sac2006-hhc=20

as part of=20

The 21st ACM Symposium on Applied Computing
http://www.acm.org/conferences/sac/sac2006=20

Zakaria
```

Figure 3.2: Snapshot of editing questionable data of a CFP

3.4 Environment used for the development of the System

- Python 2.4.1 developed in the Stani's Python editor 0.8.2 [11].
- Web services hosted over Apache Tomcat. Developed in eclipse 3.1 using Web services tools [12].
- MySQL 5.0 to host the database that holds the data.

3.5 Assumptions made for the system

These are the assumptions made by the system:

- The inbox receiving the CFP e-mails is on an IMAP server. The CFP e-mails are in the INBOX folder of the mailbox. The mailbox has 3 pre-existing folders to which the parser can redirect the categorized CFP e-mails. These folders are listed in Chapter 5 in Section 5.3.
- The e-mails that we obtain are in the RFC822 format[13].
- When the Web service is up and running, the exposed WSDL is accessible by CeParse.
- The database is created on the server hosting CReWS and CReWS can authenticate to the

MySQL database to read and write onto it. The database used is cfp_db and its schema is listed in Appendix B

- The CFP e-mails that are accessed by CeParse will have some parts of the data that CeParse is trying to look for. An ideal example of such an e-mail is listed in listing 3.1.

Listing 3.1: An example of an ideal CFP e-mail

```
===========================================================================
                              UAI 2006
          22nd Conference on Uncertainty in Artificial Intelligence
                     July 13th - July 16th, 2006
                         Cambridge, MA, USA
                   http://www.ics.uci.edu/~csp/uai2006
===========================================================================

IMPORTANT DATES:

Thursday, March 9, 2006, noon EST      Abstract submission
 Thursday,March 16, 2006, noon EST     Full paper submission
Thursday, March16, 2006, noon EST     Student paper certification
Friday, May 5,2006, noon EST          Author notification Sunday, May
21, 2006,noon EST Camera ready copy of accepted papers

FURTHER INFORMATION:
 http://www.ics.uci.edu/~csp/uai2006
 pc-chairs@ics.uci.edu
```

The CFP data extracted from this e-mail by CeParse is :

Event Name: UAI 2006
Event Type: Conference
Submission Deadline: 2006-03-16
Start Date: 2006-07-13

End Date: 2006-07-16

Venue: MA

Keywords: artificial intelligence

URL: http://www.ics.uci.edu/∼csp/uai2006

Chapter 4

CReWS

In this chapter, we describe the Web service based core component of our system, the CFP Repository Web Service or CReWS. This is essentially the repository of CFP data that accomplishes two important functions:

- Accepting data from various clients.
- Displaying and editing CFP data via the GI.

Figure 4.1 illustrates these functions and in the following sections, we explain each function in detail.

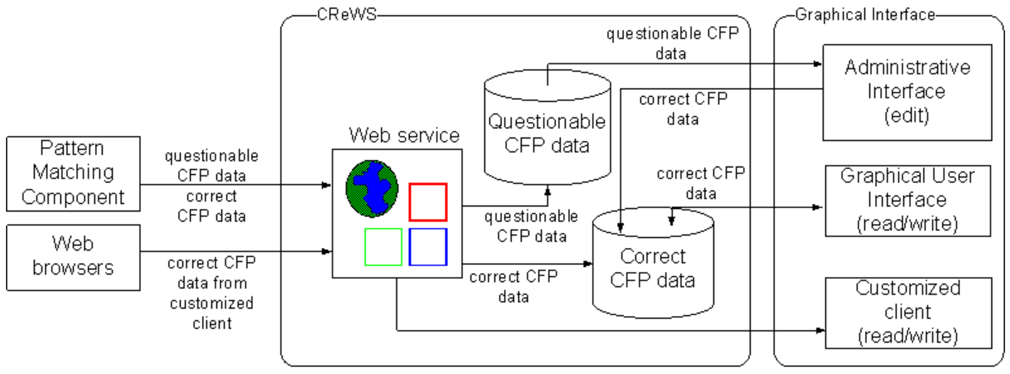


Figure 4.1: CReWS

4.1 Accepting CFP data from various clients

As mentioned earlier in Chapter 3, CReWS provides two ways of reading and writing the data in the CFP repositories.

1. JSP/Servlet implementation
2. Web services

Both these mechanisms are now described in detail in the following sections.

4.1.1 Accept CFP data using JSP/Servlet interface

CReWS provides a JSP interface to accept the CFP data. This JSP validates and forwards the entered information to a servlet. This servlet in turn makes a local call to the java implementation, while passing all the values of the CFP data. The java implementation checks the entered values for duplicity and stores them in the correct repository. Any web browser client can access the JSP over the internet, obtain the CFP data from the user and submit the data to the central repository. On successful submission, the user gets a confirmation and the data is stored in the central repository and is available for the world to view in the graphical user interface. This method of submitting CFP data can be easily adopted by any event host, since it is easy to use and requires no extra implementation. This method of a client invoking the CReWS via a JSP is exemplified by the GUI component of the GI in our system.

4.1.2 Accept CFP data using Web services

CReWS exposes two Web services in the WSDL that can be invoked to forward CFP data to the data repositories. They are

1. Write CFP data to the correct repository.
2. Write CFP data to the questionable repository.

In our implementation, CeParse invokes the first service to send correctly parsed data from the CFP e-mails to the correct repository. It invokes the second service to forward questionable CFP data to the questionable repository. While storing the data in the correct repository, it also performs a duplicity check and discards duplicates. It also incorporates CFP updates by comparing similar CFP entries in the database and overwriting them.

This WSDL can be invoked by anyone wishing to develop their own customized clients to send CFP data to the correct repository. We now summarize the way in which the data repositories are enabled to work with Web services in CReWS in the following steps.

- Java implementation for editing/writing to the Data Repositories.

  As per our requirements, we need a data repository to store the CFP data. We define a database in MySQL for the same purpose. The database schema is listed in Appendix B. It has two tables, one for the correct CFP data and the other for the questionable CFP data. Different Java APIs are provided for reading and writing data to both.

- Web service enablement of Java implementation via Eclipse Tooling

  Eclipse tooling provides a WTP (Web Tools Platform) plug-in for the Eclipse platform which extends the Eclipse platform with tools for developing J2EE web applications including Web services. Using this plug-in, Web services artifacts and skeletons are generated on the server side for the java implementation. These artifacts include serializers/deserializers for complex data types, XML bindings, WSDL etc.

- Exposed WSDL

  The WSDL that is generated from the tooling is exposed to the external world. A link to the WSDL is provided to clients who wish to access it. This WSDL exposes a named set of operations called *portType*. These operations can be invoked by Web services clients. For example, our WSDL exposes the following operations for portType "WritetoDB" : *addThistocorrectDB* and *addThistotroubleDB*. Operation addThistocorrectBD adds the parsed CFP data to the correct repository. Operation addThistotroubleDB adds the questionable data to the questionable repository.

4.2 Displaying CFP data via the Graphical Interface

As explained in the above section, CReWS uses two methods to accept CFP data into the repository. Similarly, CReWS can use both the methods to display the CFP data present in the repositories.

4.2.1 Display CFP data using JSP/Servlet Interface

Using this functionality, CReWS assists both components of the GI in displaying the data in browsers.

- The GUI can display all the CFP data for public viewing. It interacts with CReWS using JSPs and servlets[23], to read data from the correct repository. The displayed data can be sorted on any field and keywords can be used to search for CFPs. The listing also provides RSS feeds.

- The GAI lists all the CFPs in the questionable repository. If there is a similar CFP in the correct repository, it lists the similar CFP in the correct repository along with the questionable one. The administrator can edit the questionable CFP data and can move the edited CFP data to the correct CFP repository. The administrator also has the option of discarding the doubtful CFP by deleting it. CReWS checks for updates and duplicates in the existing data while adding the edited data.

4.2.2 Display CFP data using Web services

CReWS exposes Web service in the WSDL that can be used to read information from the repositories. They are:

1. Read CFP data from the correct repository.

2. Read CFP data from the questionable repository.

These Web service can be invoked by any client wishing to customize the listing of CFPs. They can also use the list of CFPs as an input to another application that needs CFP data. We explained the steps to enable the correct repository as a Web service, in Section 4.1.2, above. The reading APIs for the correct and questionable repositories are exposed in the same fashion. The portType exposed for reading data from the CFP repositories is “ReadfromDB”. The operations that is exposes are: *readcorrectDBcontentsinanArray*, *readthiscorrectCFP*, *readtroubleDBcontentsinanArray*, and *readthistroubleCFP*. The first two operations read CFPs from the correct repository while the last two read CFPs from the questionable repository.

We currently do not use this functionality in our implementation. But it can be adopted by anybody who wishes to develop their own customized Web services clients to obtain data from the questionable or correct repository.

Chapter 5

CeParse

The systematic condensation of CFP data is the driving force behind developing this implementation. We achieve this condensation using the parser, CeParse (CFP Email Parser), which has a pattern matching component. It accomplishes the complete process of obtaining the CFP e-mail to invoking the Web services in four key steps as illustrated in Figure 5.1. They are:

1. Obtaining CFP e-mail content
2. Extracting the CFP data
3. Classifying the CFP e-mail
4. Forwarding the CFP data

Each step is described in the following sections. Section 5.1 explains how CeParse obtains the CFP e-mail content from the mail server. Section 5.2 explains how the data is extracted from the e-mails. It also explains the algorithm used to parse the data along with the patterns that guide the algorithm. We also list the steps of the algorithm CeParse in Listing 1. The classification of the e-mails on the basis of pre-defined rules called classification patterns, is mentioned in Section 5.3. Finally, Section 5.4 explains how the extracted data is forwarded to the central or questionable repositories by invoking the corresponding Web services.

5.1 Obtaining CFP e-mail content

CeParse can access e-mails on a mailbox on a mail server. For our implementation, we work with the NC State's IMAP mail server. When CeParse is invoked, it prompts the user to

Algorithm 1 CeParse()

```
1: Authenticate credentials on IMAP server.
2: while (there exists e-mail in inbox) do
3:   Obtain body and subject of e-mail
4:   if (subject is not null) then
5:     extract venue from subject
6:     extract startdate and enddate from subject
7:     extract submissiondeadline from subject
8:     extract event name from subject
9:   end if
10:  if (body is not null) then
11:    extract url, keywords and eventtype from body
12:    if (submissiondeadline is null) then
13:      extract submissiondeadline from body
14:    end if
15:    if (venue is null) then
16:      extract venue from body
17:    end if
18:    if (startdate and enddate are null) then
19:      extract startdate and enddate from body
20:    end if
21:  end if
22:  classify the e-mail and move the e-mail on the inbox to corresponding folder
23:  invoke the appropriate Web service to forward the parsed cfp data
24: end while
25: Log out of the mail server
```

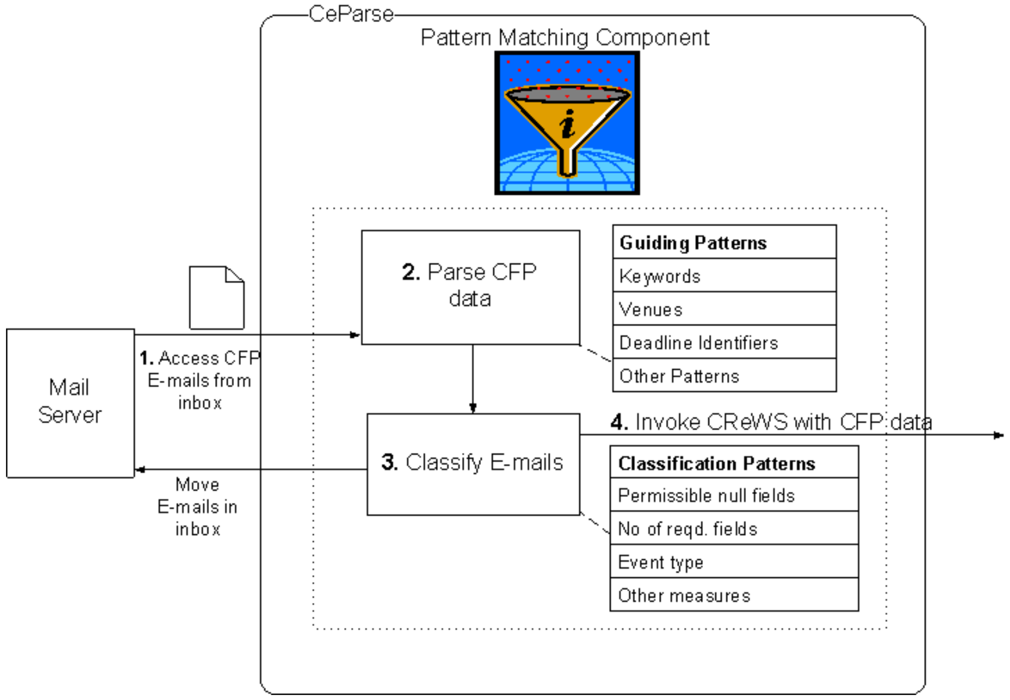


Figure 5.1: CeParse

enter the username and the password of the mailbox that is to be accessed on the IMAP server to get the CFP e-mails. It looks for e-mails that are available in the inbox folder of the mailbox. For each mail, it extracts the e-mail body and the subject (from e-mail header) into 2 different strings. It processes these strings to extract the CFP data as explained in the following section. This step of obtaining these strings is indicated in steps 1, 2 and 3 of Algorithm 1.

5.2 Extracting the CFP data

After obtaining the subject and the e-mail content from the mail server into string variables, the parser follows steps 4 to 21 of Algorithm 1 to look for the required fields of the data in each mail. The algorithm is helped by guiding patterns. We first list the fields of the CFP data that the algorithm aims to extract.

5.2.1 Extracted fields of the CFP data

The fields of the data are:

1. Event Name
2. URL
3. Submission Deadline
4. Start date of the event
5. End date of the event
6. Event Type
7. Areas of research
8. Venue
9. Actual content of the e-mail
10. Actual subject of the e-mail

5.2.2 Steps to extract each field of the CFP data

It looks for each field of the data as explained below.

- Venue

  CeParse searches for pre-defined venues, which are a part of a list of cites and the countries they are in. It searches for these venue in both the strings that are obtained from:

  - The subject of the e-mail.
  - The body of the e-mail.

  First, it looks for venues in the original subject of the e-mail as depicted in Step 5 of Algorithm 1. If it does not find any venue in the subject, it looks for the venue in the body of the e-mail, (i.e., Step 16). It looks for the event name (extracted in Step 8, discussed later) in the body of the e-mail. If the event name is found in the mail, then the parser looks for the venue in a *search string*. The search string is comprised of fifty characters immediately before and fifty characters after the event name. If it fails to find a venue in this search string, in each step it increases the length of the search string, that is centered around the event name, until it finds a venue or exhausts the e-mail body, whichever comes earlier. If it is still unable to find the venue, then the parser breaks the event name into a list comprising of all the words in the event name minus the prepositions. The parser then repeats the process of searching for a venue around each word in the list of words till it finds a venue or exhausts the e-mail body.

For example, a subject such as "[*UAI*] [*Mlnet*] CALL FOR PAPERS: ICML 2005, Bonn, Germany, 7-11 August 2005 (Submission deadline: 8th March 2005)" would yield the venue Bonn, Germany.

- Start and End date of the event

  The parser looks for a duration of time for e.g., 3rd September - 7th Sept. 06 in the subject string. This is Step 6 in Algorithm 1. The patterns for these durations or date ranges are listed in Appendix A. If it finds a date range in the subject, then that date range is considered to be the duration of the event, i.e. the start date and end dates of the event are determined. If the subject yields no results for a date range, then the parser looks for date ranges in the e-mail body (Step 19 in Algorithm 1). If a range is not found, CeParse looks for the occurrence of a single date in the body of the e-mail. This date is considered to be the start date of the event, while the end date is set to be null.

  Searching for dates involves searching for all the possible date patterns. The patterns that we search for, are mentioned in Appendix A. The found date is converted to the ISO format of yyyy-mm-dd, for e.g., March 24, 2006 will be converted to 2006-03-24.

  For example, a subject such as "[*UAI*] [*Mlnet*] CALL FOR PAPERS: ICML 2005, Bonn, Germany, 7-11 August 2005 (Submission deadline: 8th March 2005)" would yield the start date as 2005-08-07 and end date as 2005-08-11.

- Event Name

  The parser looks for the name of the event in the subject of the e-mail, that it has obtained from the mail server (Step 8 in Algorithm 1). To obtain the event name from the subject, it has to strip off certain words and characters from the subject. The key words that have to be removed are values of venues, conference dates or deadlines. We have already described the search for venues and dates earlier in this section. These words are removed from the subject to obtain the event name. Some words that also have to be removed include words that describe the content of the CFP e-mail. Here are the descriptive words that we remove from the subject to extract the event name:

  - 1st/2nd/3rd/4th/First/Second/Last/Final/Updated/Preliminary Call for
  - Joint/Preliminary Call for
  - Call for Participation/Paper
  - Paper/Submission/details extended due/on/by/to
  - Papers due extended to/by

- Extended/Early Deadline/Registration until/to
- Special Issue
- CFPs/Announcement of/for/in
- Re:/Fwd:/Fw:/Reminder/Extension/Approaching/is coming
- [$nameofmailinglists$]

Along with the formats of the dates mentioned in Appendix A, dates with formats mentioned here are also extracted to obtain the remaining title of the event.

- mon_dd e.g. Sep 2.
- dd_mon e.g. 15 February.
- mon_dd_range e.g. August 8th - Aug 10.
- dd_mon_range e.g. 1 Jan - 3 January.

These formats of dates do not give us much insight into judging the dates of the conference, as they do not have a year component. So, unless we compare them with the current timestamp of the system in a way to estimate the value of their year component,these dates are of not much use to us.

For example, a subject such as "[UAI] [$Mlnet$] CALL FOR PAPERS: ICML 2005, Bonn, Germany, 7-11 August 2005 (Submission deadline: 8th March 2005)" would yield the event name of "ICML 2005".

- Submission Deadline

  The parser first looks for a submission deadline in the original subject string of the e-mail (Step 7 in Algorithm 1). If it does not find one, then, the parser looks for the deadline in the e-mail body string (Step 13 in Algorithm 1). To look for the submission deadline in a string, the parser first looks for some pre-defined keywords in the string. These are specified in the submission deadline guiding pattern and we list these keywords below. If it finds a keyword, it first obtains a search string that is made up of two lines above and two lines following the found keyword (by lines we mean sentences separated by new line characters). It then looks for dates in this search string. To look for the dates, it follows the same guidelines as we explained above, to look for single dates. The list of keywords in order that serves as a guiding pattern is:

  - Paper submission deadline
  - Submission of paper

- Paper due
- Submission deadline
- Submission
- Deadline

For example, a subject such as "[*UAI*] [*Mlnet*] CALL FOR PAPERS: ICML 2005, Bonn, Germany, 7-11 August 2005 (Submission deadline: 8th March 2005)" would yield the submission deadline of 2005-03-08.

Listing 5.1: An example of an CFP e-mail body

```
>>>> PLEASE ACCEPT OUR APOLOGIES IF YOU RECEIVE MULTIPLE COPIES <<<<

Dear Colleague,

you might be interested in the following call for papers and web
site:

http://www.doc.ic.ac.uk/~ue/DALT-2005/


Best regards,
Matteo, Ulle, Andrea, and Paolo

-- Dr. Matteo Baldoni Dipartimento di Informatica        |
Universita' degli Studi di Torino | Tel. +39 011 6706756 C.so
Svizzera, 185                     | Fax. +39 011 751603 I-10149 Torino
(Italy)            | URL http://www.di.unito.it/~baldoni
--------------------------------------------------------------------------------
Please avoid sending me Word or PowerPoint attachments. See
http://www.fsf.org/philosophy/no-word-attachments.html
--------------------------------------------------------------------------------
```

- URL

  The parser looks for the first URL that it can find in the e-mail body based on the following guidelines (Step 11 in Algorithm 1):

  - It looks for the first occurrence of ‘www.’ in the body of the e-mail.
  - If found, it checks if ‘www.’ is pre-pended by ‘http://’, or ‘https://’. It it finds this is true, then it assumes that the URL string starts from the the position of ‘http://’ or ‘https://’ else it assumes the URL begins with ‘www.’. If it does not find any occurrence of ‘www.’ , it looks for the occurrence of ‘http://’ or ‘https://’ and if found, it assumes that to be the start of the string. If both these cases are not found, CeParse concludes that there is no URL present in the CFP e-mail.
  - If it finds either of the cases above, it knows where the URL string starts from, it looks for the end of the URL by searching for a space or a new line or the end of file. This is considered to be the end of the string. And hence, the entire string from the found start of string to found end of string is assumed to be the URL string.
  - Now the obtained string is cleaned off any trailing spaces or any ‘.’ or any ‘)’ that might have got picked up as a result of looking till the end of line.

  Thus the URL is obtained from the body of the CFP e-mail.

  For example, Listing 5.1 yields the URL “ http://www.doc.ic.ac.uk/∼ue/DALT-2005/”.

- Keywords or areas of research

  CeParse is provided with a pre-defined list of keywords that serves as a guiding pattern to search for areas of research in the CFP e-mails (Step 11 in Algorithm 1). The parser searches for these keywords and extracts a list of all the keywords that it finds in the e-mail. The current list of keywords that it looks for are:

  - Agents
  - Ambient Intelligence
  - Artificial Intelligence
  - Complex Systems
  - Constraint satisfaction
  - Data Mining
  - E-commerce
  - Formal Methods

  - Graphics
  - Grid Computing
  - Knowledge Discovery
  - Logic Programming
  - Machine Learning
  - Modalities
  - Multi-Agent Systems
  - Optimization
  - Pervasive Computing
  - Services Oriented Computing
  - Semantic Web
  - Ubiquitous Computing
  - Web services

  For example, Listing 5.1 will not yield any keywords as none of the words match our list of keywords.

- Event Type

  The parser categorizes each CFP as a call for a conference, workshop or journal (Step 11 in Algorithm 1). To determine this category, it counts the occurrences of three corresponding words in the body of the e-mail. The words that is looks for are:

  - Conference
  - Workshop
  - Journal

  The mail is categorized as a call for a conference, workshop or journal, depending on the word that is encountered the maximum number of times in the e-mail. If none of these words are found or if either of them are encountered an equal number of times as another in the e-mail, the e-mail is categorized in the 'doubtful' category.

  For example, the e-mail in listing 5.1 is categorized in the 'doubtful' category.

- Actual content of the e-mail

This is the original content of the e-mail truncated to the first 5000 characters. This is visible to the users in the Central repository in case the person viewing the listing wanted to see the actual content of the e-mail.

The example for this is the entire e-mail in Listing 5.1.

- Actual subject of the e-mail

  This is the original subject of the e-mail. This is passed to the Web service in case the administrator needs to look up any mail mapped to the original content.

  An example is "DALT 2005: second call for papers", the subject that accompanied the e-mail in Listing 5.1.

5.3 Classifying the CFP e-mail

The values of the extracted data lead to the classification of the CFP e-mails into three categories (Step 22 in Algorithm 1). They are:

1. Correctly parsed e-mails (yielding the CFP data).
2. Questionable e-mails (yielding questionable data).
3. Unclassifiable e-mails (yielding no data).

A null value of any of the following fields classifies the e-mail as a questionable e-mail:

- Event Name
- Submission Deadline
- Start Date
- URL

Further, if the determined Event Type is 'doubtful', the e-mail is classified as a questionable e-mail. The fields listed above can have null values if the parser is unable to find their values in the body of the e-mail.

In case, the e-mail is in a format that cannot be decoded by the parser at all, then it is marked as an Unclassifiable e-mail.

These e-mails are then moved to folders with the same names as the classification on the mailbox, in case the administrator wants to view the messages on the mail server later on (Step 22 in Algorithm 1). The Python library that is used to interact with the IMAP server is imaplib[14]. It permits moving e-mail messages to various folders. The folders that have to be created on the server beforehand are:

1. Correctly_parsed_cfps
2. Questionable_cfps
3. Unclassifiable_cfps

5.4 Forwarding the CFP data

Based on the category of the e-mail, the respective Web service is invoked to forward the extracted CFP data to CReWS (Step 23 in Algorithm 1). CeParse uses SOAPy to access the WSDL file[10] exposed by CReWS and invoke the operations defined in the WSDL. Here are the actions that are invoked by CeParse in each case:

1. Correctly parsed e-mail

   Action : CeParse invokes the CReWS Web service to put the data in the correct repository. It not only sends all the the fields of the data that it is able to parse but it also sends the complete e-mail body and original subject.

2. Questionable CFP

   Action : CeParse invokes the CReWS Web service to send the questionable CFP data to the questionable repository. It sends all the CFP questionable data that it is able to parse along with the e-mail body and subject. A null value is stored in the database for all those values that the parser is unable to find. The GAI provides the option to the administrator to manually check the e-mail and correct/update the questionable fields and hence store them in the correct CFP repository.

3. Unclassified e-mails

   If an e-mail cannot be parsed by CeParse, the details or null values are not sent to CReWS at all. This e-mail is moved to a folder named Unclassifiable in the mailbox for the administrator to review. Hence no Web service is invoked here.

CeParse then logs out of the e-mail server (Step 25 in Algorithm 1). Please refer to chapter 4 for details on the working of CReWS.

5.5 Intuition behind the Guiding and Classification Patterns

CeParse aims at parsing CFP e-mails that are composed by human beings. These e-mails do not follow a standard template, which implies that we do not have a fixed idea of the kind of input to expect for CeParse. Hence, CeParse has to be ready to accept a wide variety of e-mails. The rules and guiding patterns that we have listed in this chapter were developed while working with a set of about six hundred e-mails. Looking at the general formats that these e-mails seem to have, we picked up the patterns that could be most effective. A prime example of such a guiding pattern development is the development of the submission deadline pattern. As we mention above in section 5.2, the keywords that we look for in the subject and the e-mail body while searching for the submission deadlines are:

1. Paper submission deadline
2. Submission of paper
3. Paper due
4. Submission deadline
5. Submission
6. Deadline

In this case when we looked for the submission deadline in the line containing the first 3 keywords, we were able to correctly extract the submission deadline for 23 out of 29 e-mails. Adding the remaining 3 to the keywords list increased the right extractions to 25. If we searched for the submission deadline around these keywords in the search string that spanned 2 lines above and below the keyword, our accuracy improved to 100% and CeParse was able to extract 29 out of 29 submission deadlines correctly.

Section 5.2 lists the guiding patterns that we use to extract the titles of the conferences, workshops and journals. We obtained this list of patterns by parsing the set of the same 600 CFP e-mails mentioned above. We observed the results of our guiding pattern list on the extracted title. By observing the results, we added new words to the pattern list to let it evolve to its current state.

The venue list that we use is a list of cities and their countries. It is not exhaustive as we keep seeing new venues in the new CFPs. And this list has to grow to encompass all the possible venues without a complete assurance that each venue will be found in each CFP e-mail.

Chapter 6

System Evaluation

We evaluate the working of our system in this chapter. Section 6.1 lists statistics that let us judge the performance of the system. Section 6.2 discusses the problems encountered while evaluating the system. Section 6.3 lists the limitations of the system.

We show the accuracy of the parser by listing figures that were generated upon testing the parser with 150 test e-mails. These e-mails are independent of the set of 600 e-mails that was used to develop CeParse. As we explained in Chapter5 some mails are classified into the 'doubtful' category. In our test set, there were 3 such e-mails. They were in formats that could not be parsed by CeParse. Hence, the figures that we show here are for a total of 147 parsed e-mails. So the accuracy to parse e-mails and post them to the Web service is at 98%. Here is a summary of the performance of CeParse for each field.

Table 6.1: Summary of CFP data extraction

Field	**Correct Extraction**
Event Name	78%
URL	97%
Submission Deadline	80%
Start Date	77%
End Date	88%
Event Type	91%
Keywords	100%
Venue	23%
Questionable E-mails	73%

6.1 Performance of implemented System

The following tables can be interpreted in this way: The first row represents count of fields when the data was actually present and the second row lists the counts when the data was absent. The first column signifies numbers of fields that CeParse was able to extract correctly, the second column signifies the number of values that we did not find while the third mentions the number of values that we found incorrectly.

Table 6.2 shows that CeParse can parse the name of the event with an accuracy of about 78%. Considering that the name is extracted by cleaning the subject of the e-mail, this can be improved by observing the type of words and strings that people add to the CFP e-mails. However, there is an occasional e-mail which has a subject that gets completely deleted while trying to delete the extra characters. One such subject is: "[UAI]Call for Papers". In this case, after the subject has been cleaned off the characters and words listed in the guiding patterns, the extracted title is empty. So in this case, we classify such a CFP as a questionable e-mail.

Table 6.2: Statistics for extraction of Event Name

x	**Found Correct**	**Not Found**	**Found Incorrect**
Present	114	0	29
Absent	0	4	

Table 6.3 shows that the URL is parsed from the CFP e-mails with an accuracy of 97%. CeParse uses the first URL that is encounters in the e-mail as the value for the URL field of the data. At times the CFPs are forwarded by people and their signatures are pre-pended to the e-mail body. In situations like that, the parser picks up the incorrect URL.

Table 6.3: Statistics for extraction of URL

x	**Found Correct**	**Not Found**	**Found Incorrect**
Present	142	0	5
Absent	0	0	

Table 6.4 shows that the submission deadline is parsed with an accuracy of about 80%. The submission deadline that is in the vicinity of words listed in the submission deadline guiding patterns (listed in Chapter 5) is currently parsed. At times e-mails indicate deadlines of others events as well in their e-mail body. In cases like that the parser at time picks up incorrect dates. Or the format of the submission deadline date might be such that it gets ignored by our list of permissible date formats as listed in Appendix A.

Table 6.4: Statistics for extraction of Submission Deadline

x	**Found Correct**	**Not Found**	**Found Incorrect**
Present	112	10	13
Absent	7	5	

Table 6.5 shows that the start date is parsed with an accuracy of 77% and Table 6.6 shows that the end date is parsed with an accuracy of 88%. For the start date and end date, we look for the date ranges and assume the start date of the range to be the start date of the event. The end date of the date range is considered to be end date of the event. This assumption can go wrong for both the dates when the e-mail also mentions another range in the CFP e-mail before the correct one. Also when looking for the start date, we look for single dates as well in case a date range is not found in the e-mail. Looking for the single date might not help a lot, as indicated by the figures that the start date accuracy is lower than that of the end date.

Table 6.5: Statistics for extraction of Start Date

x	**Found Correct**	**Not Found**	**Found Incorrect**
Present	112	1	20
Absent	1	13	

Table 6.6: Statistics for extraction of End Date

x	**Found Correct**	**Not Found**	**Found Incorrect**
Present	89	15	1
Absent	41	1	

Table 6.7 shows that the type of the event is estimated correctly with an accuracy of 91%. Most of the event that were judged incorrectly belonged to a category od special issues. This is something that we do not look for in the CFP e-mails in the current implementation.

Table 6.8 shows a completely accurate parsing of keywords. This is enabled by an exhaustive list of keywords that are searched for in the e-mail. This list can grow depending on the areas of interest that we want to search for. Currently the keywords in it helped to find at least one keyword in each mail of our test set.

Table 6.9 shows that Venue is the field that yields the lowest accuracy. Venues are searched for based on a list of cities and countries they are in. This is something that needs much more research. As the list of places can always keep growing and yet there will be more places

Table 6.7: Statistics for extraction of Event Type

x	Found Correct	Not Found	Found Incorrect
Present	134	0	6
Absent	0	7	

Table 6.8: Statistics for extraction of Keywords

x	Found Correct	Not Found	Found Incorrect
Present	147	0	0
Absent	0	0	

where conferences are held. Further, looking for the short forms of states can be daunting task, as the short forms can occur in the e-mails several times. For e.g., IN can occur in several sentences such as “This occurs *in* this sentence”. In this case this is not a state as per the context but the parser clearly lacks the ability to understand such differences.

Table 6.9: Statistics for extraction of Venue

x	Found Correct	Not Found	Found Incorrect
Present	34	0	113
Absent	0	0	

Table 6.10 lists the numbers that show the detection of a questionable e-mail. We see that we can detect questionable e-mails with an accuracy of 73%. An e-mail is questionable if it has a null value for any one of the following fields: event name, start date, submission deadline, URL. A doubtful value for the event type also marks it as a questionable e-mail. So effectiveness of a questionable e-mail can be improved only if the effectiveness of all these fields can be improved.

6.2 Problems with evaluation

As the system parses e-mails that are composed by human input rather then by any automated process, checking the extraction result entails checking the extracted data for each CFP manually. This is a tedious and time consuming process. However, a test set can be generated while parsing the CFP e-mails and the result of parser can be checked with that test set. This is the approach that we follow. The problem of generating new test remains. And that has to be accomplished by reading each e-mail manually.

Table 6.10: Statistics for detection of questionable e-mails

x	Found Correct	Not Found	Found Incorrect
Present	36	16	15
Absent	72	7	

6.3 Limitations of the system

It can not be 100% accurate as the human written e-mails do not have a fixed template. Also, as the e-mails are written by humans, they are themselves prone to errors. If the input to this system is erroneous or ambiguous, it cannot extract the information from the CFP e-mails. Further, non-ascii characters are handled by Python if it knows exactly the type of encoding that they follow. However, the variety of e-mails received is great and a new variety can always be encountered that Python does not check for.

6.3.1 E-mails with erroneous data formats

Here in listing 6.1 we list the segment of a CFP which has an erroneous input for the parser in terms of the start and end date of the conference as visible in line 8 of this listing.

Listing 6.1: An erroneous CFP e-mail segment

```
1  The 2nd International Workshop on Web and Mobile Information Systems
2                               (WAMIS'06)
3 http://euler.acadiau.ca/~eshakshu/CIDS/WAMIS06/index.htm
4
5 in conjunction with the IEEE 20th International Conference on
6        Advanced Information Networking and Applications (AINA2006)
7            http://www.takilab.k.dendai.ac.jp/conf/aina/2006/
8               April 18 - April, 2006, Vienna, Austria
```

6.3.2 E-mails with ambiguous data

Several input CFPs are not erroneous but they do not serve as the correct input for our parser. An example of such an input it listed in listing 6.2. The date of the conference is to be determined (line 7 of the list). But this can be understood by a person reading this e-mail, while it defeats CeParse.

6.3.3 E-mails with non-ascii characters

Non Ascii characters in the e-mails are ignored. Python supports unicode characters while manipulating strings. However when the data is transferred over to the CFP repository, the non-ascii characters cannot be handled by the output stream and exceptions are raised. So in order to handle the CFP body while transferring the data, we ignore the non ascii characters. One such example is shown in listing 6.3, line 7. i.e. MÃílaga.

Listing 6.2: An ambiguous CFP e-mail segment

```
1 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
2            Semantic Web Personalization Workshop
3            @ 3rd European Semantic Web Conference
4
5         http://www.kbs.uni-hannover.de/~henze/swp06/
6                    June 11 or 12, 2006 (tbc)
7                      Budva (Montenegro)
8 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Listing 6.3: E-mail segment containing non-ascii characters

```
1 2ND CALL FOR PAPERS (20 February, 2006) Resource-Scarce Language
2 Engineering http://altiplano.emich.edu/resource_scarce/ 31 July - 4
3 August, 2006
4
5 organized as part of the European Summer School on Logic, Language
6 and Information ESSLLI 2006 http://esslli2006.lcc.uma.es/ 31 July -
7 11 August, 2006 in M laga
```

6.4 Contributions and Reusability

This work has created a system to parse and manage calls for papers that has a high degree of automation associated with it. Manual intervention for creating a centralized listing of CFPs has been reduced significantly. And manual intervention is still required as the parser is not 100% accurate in its extraction of CFP meta-data. As the accuracy of the parser would rise, the manual intervention can reduce.

Also, human parsable e-mails cannot be parsed on the basis of one single format. Instead, several permutations and combinations have to be tried to reach at the most promising results. Searching for the names of venues is considered to be an example of a classic problem of named entity recognition[15]. Our way of searching for the venue in an e-mail is a simplistic attempt to search for the venues where the conferences are held.

There are several features of this developed system that can be re-used by other systems being developed. For e.g., the pattern matching techniques developed by this system to look for various patterns in the CFP e-mails can be used for applications as well. This parser extracts information out of e-mails that are typed by humans and have no formats what so ever attached with them, so a number of patterns were developed to search for the desired fields. For e.g., CeParse searches for several date formats in order to be able to locate all the dates in the e-mail. So these patterns of searching for various date formats can be used by any other application that has to search for dates. For e.g., searching for dates in news articles or web pages. Similarly, this application can be adapted by any other academic or industrial community to parse their own CFPs, with the alteration of changing keywords. The patterns that we have identified to search for URLs can be further used by anyone wishing to search for URLs. For e.g., for a person creating a script to search for all the URLs present in text files on his server.

Chapter 7

Future Work

The opportunity for future enhancement and additions to this system is tremendous. We examined the accuracy figures in the last chapter. The performance of CeParse can be improved to a great extent. The features of CReWS can be tapped to add to the existing features of this system. We list enhancement areas for both the parts of the system here.

Some features that can enhance CReWS are listed here.

- Archive the CFP-data present in the database.
- The duplicity check that the correct repository performs while entering in CFP data, can be enhanced to looking for a similarity factor, while gauging if the e-mail is a duplicate or a new entry.
- It can be configured to send out CFP digests to the people who subscribe to them.
- It can set cookies on the machines of people to remember user preferences when the GUI is invoked by any user. In this case the users will not have to search for their relevant conferences each time they access the CFP central listing.

We list some of the possible enhancements for CeParse here.

- Improve accuracy of parsing the CFP data.
  - The first extraction that can be made more accurate is the Venue field. We can incorporate an exhaustive list of venues that comprises of all the big cities in the world and university towns. We can search for the cities only, since extracting the city would be a task which could be achieved with more accuracy.

  - The second is that of the URL. We can check for the correctness of the extracted URL by pinging it over a period of time to see if it is live. Also, emphasis on looking for the name of the event or the event acronym in the URL in case of multiple URLs could help.
  - Event names can be searched for in the body of the e-mail. We can search for the common occurring conference names or look for an acronym in the URL such as IEEE, AAMAS, ACM, SIG. This acronyms can be searched for in the body of the e-mail.
  - Several other fields can be extracted from the e-mails, namely Registration dates, Acknowledge dates, Abstract submission deadlines.
  - The number of categories identified by CeParse can be increased by searching for calls for game competitions and by looking for CFPs that announce multiple tracks.
  - Further, if CeParse has been unable to search for some fields in the e-mail, we can obtain the content of the page that the URL points to, and look for the missing information on that page.
  - A complete new avenue exists in stripping off the html formatting that accompanies the e-mail body of some CFPs.
  - As mentioned in section 6.3, currently the special characters are replaced with a space and hence ignored by CeParse. We can instead incorporate value maps for these non-ascii characters, which would map all of these special characters to pre-specified ascii representations. For e.g., Ã can be mapped to A, etc. The special characters can be identified by searching each CFP for a list of special characters. Such a list of characters can be found at online sources like the website[16].

- A support for parsing attachments can be added.
- If CeParse is unable to parse an e-mail or categorizes it in the doubtful category, the e-mail can be returned to the sender with a message asking him to send the e-mail in a format that is acceptable by CeParse.
- Guidelines for sending CFPs in the right format can be indicated on the website that hosts the listing of the CFPs.
- An interface for the parser can be incorporated where the rules and the patterns for parsing the CFPs can be specified at runtime. This would enable the parser to extract any data from the CFPs, for example, this would help change the keywords being searched for.
- The parser can apply learning algorithms to improve its effectiveness.

- CeParse can be ported to become a Web service so that it also becomes inter-operable and can access any mail server.
- Incorporate support for all non-unicode characters.

Also, currently the GUI accesses the database via JSPs and servlets that interact with the local java implementation to read and write data to the data repositories. The GUI can be web service enabled so that it invokes only the exposed functions in the WSDL to obtain/enter data to the repositories. This will ensure that there is only one entry point to enter and extract data from the system. Also, wrapper interfaces[17] can be developed to extract data from the data repositories and to specify the capability of the web accessible data sources.

Bibliography

[1] http://www.acm.org/sigs/sigecom/exchanges/announce.html.

[2] http://cfp.english.upenn.edu/.

[3] http://www.hypermail-project.org/.

[4] http://www.papersinvited.com/.

[5] http://www.csa.com/.

[6] http://www.w3.org/TR/wsa-reqs/#id2604831.

[7] http://soapy.sourceforge.net/.

[8] http://www.python.org/.

[9] http://www.regular-expressions.info/.

[10] http://www.w3.org/TR/wsdl.

[11] http://stani.be/python/spe/.

[12] http://www.eclipse.org/webtools/initial-contribution/IBM/evalGuides/WebServicesToolsEval.html.

[13] http://www.faqs.org/rfcs/rfc822.html.

[14] http://www.python.org/doc/lib/module-imaplib.html.

[15] http://www.cnts.ua.ac.be/conll2002/ner/.

[16] http://www.bbsinc.com/symbol.html.

[17] ftp://ftp.umiacs.umd.edu/pub/louiqa/BAA9709/PUB98/1CoopIS98.pdf.

[18] http://www.service-architecture.com/.

[19] http://www.w3.org/XML/.

[20] http://www.w3.org/TR/soap/.

[21] Jeffrey E. F. Friedl. *Mastering Regular Expressions.* O'Reilly, 2002.

[22] Alex Martelli. *Python in a Nutshell.* O'Reilly, first edition, 2003.

[23] Larry Brown Marty Hall. *Core Servlets and Javaserver Pages.* Prentice Hall, 2004.

[24] Peter Norton. *Begnning Python.* Wiley Publishing, 2005.

Appendices

Appendix A

CeParse Guiding Patterns - Date Formats

- Date formats for submission deadlines or a conference date, Here the delimiters can be / or . or -
  - yyyy_mm_dd e.g. 2006-08-24
  - mm_dd_yyyy e.g. 08-24-2006
  - dd_mm_yyyy e.g. 24-08-2006
  - mon_dd_yyyy_words e.g. August 8th, 2006 or Aug 08 2006
  - dd_mon_yyyy_words e.g. 8 Aug 2006
- Date formats for conference date ranges
  - mon_dd_yyyy_range e.g. Sep 2 - Sep. 7th 2006
  - dd_mon_yyyy_range e.g. 3rd September - 7th Sept. 06
  - mon_dd_yyyy_rangeovermonths e.g. Feb 28 - march 2nd 2006
  - dd_mon_yyyy_rangeovermonths e.g. 28 February - 2 March 06

Appendix B

Database Structure

The database cfp_db consists of two tables. CORRECT_CFPS and QUESTIONABLE_CFPS. The structure of both the tables is listed here.

Table B.1: Structure of table CORRECT_CFPS

Field Name	**Type**	**Length**	**Default**
CALL_ID	INTEGER	10	NOT_NULL, AUTO_INCREMENT, PRIMARY_KEY
EVENT_NAME	VARCHAR	200	NULL
EVENT_TYPE	VARCHAR	50	NULL
SUB_DL	DATE		NULL
START_DATE	DATE		NULL
END_DATE	DATE		NULL
VENUE	VARCHAR	100	NULL
KEYWORDS	VARCHAR	300	NULL
DESCR	VARCHAR	300	NULL
URL	VARCHAR	200	NULL
E-MAIL_BODY	VARCHAR	1000	NULL

Table B.2: Structure of table QUESTIONABLE_CFPS

Field Name	**Type**	**Length**	**Default**
CALL_ID	INTEGER	10	NOT_NULL, AUTO_INCREMENT, PRIMARY_KEY
EVENT_NAME	VARCHAR	200	NULL
EVENT_TYPE	VARCHAR	50	NULL
SUB_DL	DATE		NULL
START_DATE	DATE		NULL
END_DATE	DATE		NULL
VENUE	VARCHAR	100	NULL
KEYWORDS	VARCHAR	300	NULL
DESCR	VARCHAR	300	NULL
URL	VARCHAR	200	NULL
E-MAIL_BODY	VARCHAR	1000	NULL

Appendix C

Web services - An overview

C.1 Web services architecture

The Web Services Architecture as illustrated in Figure C.1, web services consist of three main components. They are the Service Registry, the Service Provider, the Service Requestor. We briefly describe them here.

- Service Registry or the UDDI (Universal Description, Discovery and Integration) that is a searchable repository of service descriptions.
- Service Provider
  - Provides applications as services
  - Publishes their services to registry
- Service Requestor
  - Client that requires a service
  - Searches the registry for available services

C.2 Underlying technology of Web services

Web services is a relatively new technology that has received wide acceptance as an important implementation of service-oriented architecture (SOA)[18]. This is because Web services provides a distributed computing approach for integrating extremely heterogeneous applications over the Internet. The Web service specifications are completely independent of programming language, operating system, and hardware to promote loose coupling between the service consumer and provider. The technology is based on open technologies such as:

- Extensible Markup Language(XML)[19]

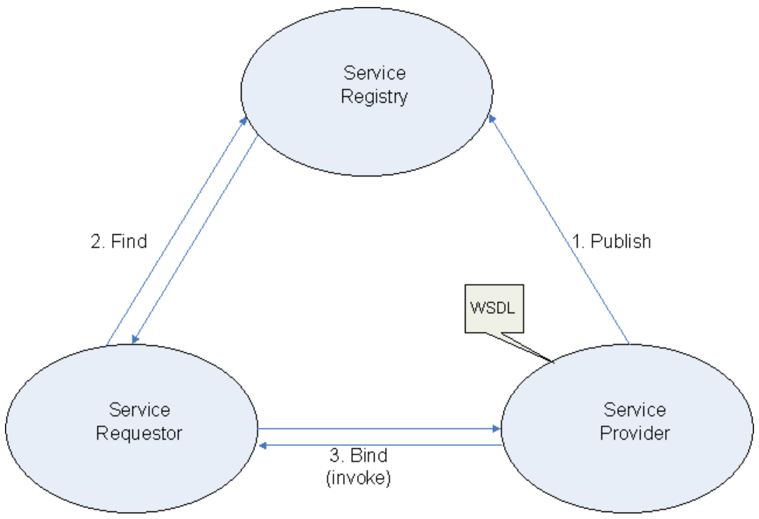


Figure C.1: Web services architecture

- Simple Object Access Protocol(SOAP)[20]
- Universal Description, Discovery and Integration (UDDI)
- Web Services Description Language (WSDL)

Using open standards provides broad inter-operability among different vendor solutions. These principles mean that companies can implement Web services without having any knowledge of the service consumers, and vice versa. This facilitates just-in-time integration and allows businesses to establish new partnership easily and dynamically.

C.3 Web services and SOA

A service is generally implemented as a course-grained, discoverable software entity that exists as a single instance and interacts with applications and other services through a loosely coupled, message-based communication model.

- Services

  Logical entities, the contracts defined by one or more published interfaces.

- Service provider

  The software entity that implements a service specification.

- Service consumer (or requestor)

The software entity that calls a service provider. Traditionally, this is termed a "client". A service consumer can be an end-user application or another service.

- Service locator

  A specific kind of service provider that acts as a registry and allows for the lookup of service provider interfaces and service locations.

Following benefits are realized with SOA pattern

- Leveraging existing assets
- Easy to integrate and manage complexity
- Reduce cost and increase re-use

SOA pattern is realized using the upcoming Web service technology where composable services are exposed for consumer to benefit from.